8-1-2021

On Some Similarities and Differences between Deep Neural Networks and Kernel Learning Machines

Eddie Pei
ep2667@rit.edu

ROCHESTER INSTITUTE OF TECHNOLOGY

THESIS

On Some Similarities and Differences between Deep Neural Networks and Kernel Learning Machines

Author:
Eddie Pei

Supervisor:
Dr. Ernest Fokoué

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science in Applied Statistics

in the

College of Science
School of Mathematical Science

August 1, 2021

Declaration of Authorship

I, Eddie PEI, declare that this thesis titled, "On Some Similarities and Differences between Deep Neural Networks and Kernel Learning Machines" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Thesis Committee

Signature Date

Dr. Ernest Fokoué
Thesis Advisor, School of Mathematical Sciences

Dr. Robert Parody
Committee Member, School of Mathematical Sciences

Dr. Linlin Chen
Committee Member, School of Mathematical Sciences

ROCHESTER INSTITUTE OF TECHNOLOGY

Abstract

Faculty Name
School of Mathematical Science

Master of Science in Applied Statistics

by Eddie PEI

This thesis presents an extensive and thorough computational comparison featuring deep neural networks and kernel learning machines and successfully establishes that on both real-life datasets and artificially simulated ones, kernel learning machines tend to be just as good as deep neural networks and quite often far better predictively. It turns out from the findings of this thesis that while deep neural networks might have worked well on tasks for which millions of observations are available, kernel learning machines just happen to be predictively better on the wide variety of tasks with the kind of sample size that one should realistically expect to have in practice.

Acknowledgements

I would like to express my sincere appreciation to my mentor and committee chair, Professor Ernest Fokoué, who has very deep knowledge, patience, and passion for research, which inspire me in all ways as a statistics outlier who entered Rochester Institute of Technology Applied Statistics program to fall in love with statistics, thanks for his training and inspirations. Moreover, it is always a pleasure to take his class which is challenging, but the rewards are amazing too. Thank him for motivating me to sequentially conduct three research projects "Graph Enhanced High Dimensional Kernel Regression,"; "How to Do Deep Neural Networks Compare Predictively Against Kernel Learning Machines?"; and "Improving the Predictive Performances of K-nearest Neighbors Learning by Efficient Variable Selection." I really appreciate the way he trained me, gave me challenges, but always there to help me; gave me enough room to explore my interests.

I would like to thank Professor Roberts Parody and Professor Linlin Chen for their acceptance of being my committee members and generous guidance during my Master's study. Professor Roberts Parody is a very easygoing and kind person, I always enjoy taking his class, and I really appreciate his help in the last couple of years! Professor Linlin Chen is a very kind person, always with a smile when you met her. It is my great honor to have you as my committee members.

I sincerely thank the Rochester Institute of Technology Applied Statistics department, thank you for all the training and helps. I am delighted to be part of it.

Contents

List of Figures

List of Tables

List of Abbreviations

DNN	**D**eep **N**eural **N**etworks
1HLNN	**S**ingle **H**idden **L**ayer **N**eural **N**etworks
SVM	**S**upport **V**ector **M**achine
GP	**G**aussian **P**rocess
MSE	**M**ean **S**quared**E**rror
ACC	**ACC**uracy
Poly	**Poly**nomial kernel
linear	**linear** kernel
CV	**C**ross **V**alidation
NTK	**N**eural **T**angent **K**ernel

List of Notations

$\mathbb{R}$: Set of real number
$\mathbb{R}^p$: p-dimension Euclidean Space
n : Sample size
p : the Number of parameters
$\mathscr{D}_n$: Sample of size n dataset
$\mathbf{x}_i$: $(x_1, x_2 \cdots x_p)^\top$
$\mathbf{X}$: matrix and data matrix
$\mathscr{H}$: Function
$\mathscr{F}$: Function space
$\mathscr{X}$:Input space
$\mathscr{Y}$: Output space
$\mathscr{Z}$: Hidden layers space
$\boldsymbol{W}$: Matrix of weights
$\boldsymbol{\theta}$: $(\theta_1, \theta_2 \cdots \theta_p)$, Parameters vector
w : $(w_1, w_2 \cdots w_p)$, weight vector
$\boldsymbol{\Theta}$: Parameter space
d_i : Dimension of input space; ie : $dim(\mathscr{X}) = d_i$

Chapter 1

Introduction

1.1 Introduction

The emergence of Data Science is an entirely new research field; the resurgence of Neural Networks is through the rise to prominence of deep learning. Deep learning has regained tremendous popularity in both practical and methodological statistical machine learning circles in recent years, owing to their remarkable good predictive performances. Some practitioners have relied on Deep Neural Networks to implicitly/indirectly insinuate that Deep Neural Networks might be the holy grail in practical statistical machine learning. Indeed, in the words of the co-inventors in the most common incarnation of Deep Neural Networks (LeCun, Bengio, and Hinton, 2015),

> *Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with various levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection, and many other domains such as drug discovery and genomics.*

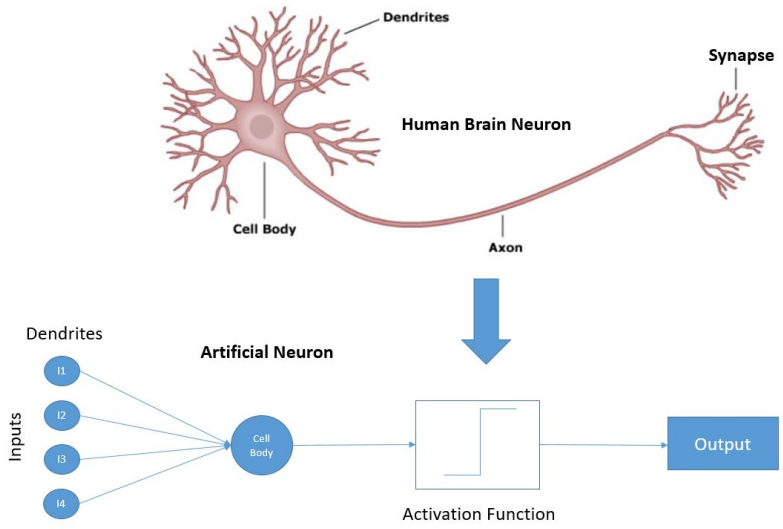


FIGURE 1.1: The comparison of Deep Neural Networks and Human Brain Cell picture sauce: http://www.mplsvpn.info/

Deep Neural Networks, as a machine learning technique, attempts to mimic the human brain. Figure 1.1 shows the similarity of Deep Neural Networks and the human brain. The brain cells, or neurons, provide the primary function of the brain.

Neurons have dendrites, which help to deliver signals between neurons. The neural is surrounded by dendrites, receives all the signs and sum them and forward them to axons. Axons help to transfer these signals to the subsequent neurons. For Deep Neural Networks, a neuron is a node with as many inputs and one output. Similar to brain networks, Neural Network consists of many neurons. It could be described that receives data at the information and provides a response. The Neural Network learns to correlate input and output signals in the learning section, and then the Neural Network begins to work. It receives more new data, and delivers the output signals based on the accumulated knowledge.

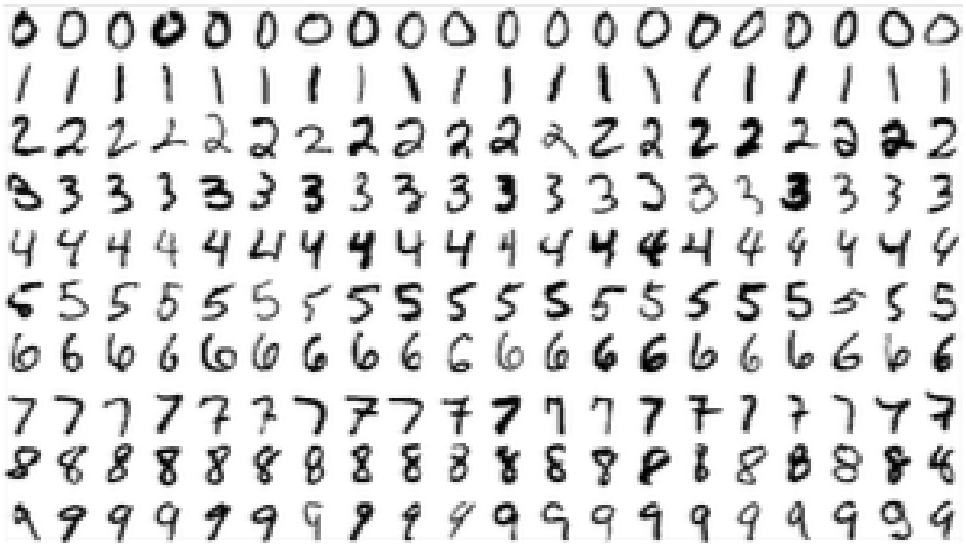


FIGURE 1.2: Example of handwritten digits from Modified National Institute of Standards and Technology database (MNIST)

Neural Networks have infused tremendous new energy into the statistical machine learning and artificial intelligence fields. In fact since their inception in the mid-1980s Deep learning helps with solving a wide variety of significant problems (Dechter, 1986; Hassoun, 1995; Fukushima, 2004). Neural Networks have enjoyed sustained tremendous success in a wide variety of machine learning tasks like image processing, audio processing, medical diagnostics, marketing research, and various science (Krizhevsky, Sutskever, and Hinton, 2012; LeCun, Bengio, and Hinton, 2015; Schmidhuber, 2015; Kawahara et al., 2017; Liu et al., 2017; Snyder et al., 2017; Tian et al., 2018; Alfaro-Almagro et al., 2018). One well-known example is the famous handwritten digit recognition machine leaning task, [shown in Figure 1.2], which is called "Modified National Institute of Standards and Technology database" short as MNIST from United States Postal Service (USPS). It is a dataset of 60,000 small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9. The task is to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9, inclusively. Deep learning could achieve a classification accuracy of above 99% on the hold-out test dataset. It has been drawing lots of attention in the machine learning field ever since Deep Neural Network's success in this entirely different task.

Deep learning can also be used for unsupervised learning. For example, word embedding, image encoding into lower or higher dimensions, extensions of PCA

like autoencoders and variational autoencoders (Amarbayasgalan, Jargalsaikhan, and Ryu, 2018 etc). Neural Networks can also be applied to reinforcement learning and deep reinforcement learning. There is now a prospect of a wide-scale deployment of DNN in safety-critical applications such as self-driving cars, which will play an essential role in the years to come. Some small-scale applications already benefit a lot in our life, such as Apple's Siri, Amazon's Echo, and Google's "hey Google".

1.2 Thesis Scope

Deep learning has indeed become an active research area in the past few years. For a Deep Neural Network machine, good performance could be expected by utilizing neurons and adjusting the weights. However, depending on the problems, the computational stages may require a long time for training the DNN model. Furthermore, when the sample size is not big enough, DNN will have an overfitting problem. Other machine learning algorithms may be more efficient for this problem, such as Support Vector Machines (SVM).

The growing number of applications of Deep Neural Networks has caused people to go as far as thinking that Deep Neural Networks should be the only machines among all. DNN is indeed a mighty machine, but does the "no free lunch" not apply to DNN? Is DNN universally superior to all other learning machines on all possible data sets?

In this thesis, we explore several fascinating topics:

- What are the similarities and dissimilarities between neural networks and kernel learning machines?
- What is the computational and methodological price Deep Neural Networks need to pay to achieve what is known as their spectacular superiority in a wide range of applications?
- Even more crucially, is it the case that Deep Neural Networks outperform Kernel machines across all possible datasets?

Throughout this thesis, we focus solely on supervised learning with equal emphasis on classification and regression. Specifically, we consider an input space $\mathscr{X}$ and an output space $\mathscr{Y}$, and we seek to build learning machines

$$f : \mathscr{X} \longrightarrow \mathscr{Y} \tag{1.1}$$

that capture the relationship between the elements of $\mathscr{X}$ and those of $\mathscr{Y}$. Typically, a huge part of the statistical machine learning process consists of choosing/selecting function space or hypothesis space $\mathscr{H}$ from which the learning machine f is drawn i.e. $f \in \mathscr{H}$, with $\mathscr{H} \subset \mathscr{Y}^{\mathscr{X}}$. Note here that $\mathscr{Y}^{\mathscr{X}}$ is the universal space of all possible functions (mappings) from $\mathscr{X}$ to $\mathscr{Y}$. Using the nomenclature from Fokoué, 2020, the learning process proceeds by using random sample

$$\mathscr{D}_n = \{(\mathbf{x}_i, y_i) \overset{iid}{\sim} p_{xy}(\mathbf{x}, y),\ \mathbf{x}_i \in \mathscr{X},\ y_i \in \mathscr{Y},\ i = 1, \cdots, n\}, \tag{1.2}$$

along with the suitably chosen function space $\mathscr{H}$ to empirically construct estimators $\widehat{f}_{\mathscr{H},n} \in \mathscr{H}$ of f. It is important to note that all datasets throughout this thesis will assumed to be random samples generated/drawn according to the joint probability density/mass function $p_{\text{xy}}(\mathbf{x}, y)$ appearing in (1.2). Now, the process of building $\widehat{f}_{\mathscr{H},n}$ from $\mathscr{D}_n$ via some optimization method or algorithm $\mathscr{A}(\mathscr{D}_n; \mathscr{H}, \Lambda)$ is the learning process. In a sense, the actual realized learning machine $\widehat{f}_{\mathscr{H},n}$ is the output of the algorithm used, so that

$$\widehat{f}_{\mathscr{H},n} := \mathscr{A}(\mathscr{D}_n; \mathscr{H}, \Lambda), \tag{1.3}$$

where Λ at this point generically denotes the collection of all hyperparameters. The theoretical framework of which $\mathscr{A}(\mathscr{D}_n; \mathscr{H}, \Lambda)$ is a manifestation, is the so-called theoretical risk minimization principle which is based of the definition of the theoretical risk functional $R(f)$ representing the population error made by a learning machine (function) $f \in \mathscr{Y}^{\mathscr{X}}$. The theoretical risk functional id defined as

$$R(f) = \mathbb{E}\Big[\mathcal{L}(Y, f(X))\Big] = \int_{\mathscr{X}\times\mathscr{Y}} \mathcal{L}(\mathbf{x}, y) p_{xy}(\mathbf{x}, y) d\mathbf{x}dy, \tag{1.4}$$

where $\mathcal{L}(.,.)$ is the loss function. The loss function plays a central role in statistical machine learning, as it helps with in the evaluation of how well the machine learning models fits the data. More rigorously, a loss function $\mathcal{L}(\cdot,\cdot)$ is a nonnegative bivariate function $\mathcal{L} : \mathscr{Y} \times \mathscr{Y} \longrightarrow \mathbb{R}_+$, such that given $a, b \in \mathscr{Y}$, the value of $\mathcal{L}(a, b)$ measures the discrepancy between a and b, or the deviance of a from b, or the loss incurred from using b in place of a. For classification learning, the natural loss function is the so-called zero One Loss defined as follows:

$$\mathcal{L}(y, f(\mathbf{x})) = \mathbb{1}(y \neq f(\mathbf{x})) = \begin{cases} 0 & \text{if} \quad y = f(\mathbf{x}), \\ 1 & \text{if} \quad y \neq f(\mathbf{x}). \end{cases} \tag{1.5}$$

For Regression learning, the most commonly used loss function is the so-called squared error loss, defined as follows.

$$\mathcal{L}(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2. \tag{1.6}$$

Although these two functions are the most commonly used in their respective contexts when it comes to evaluating learning machines, many other loss functions exist and are used for a wide variety of specific machine learning scenarios deemed more suitable. Given a suitable chosen loss function along with the risk functional defined in (1.4), one ideally wants to find the universal best function $f^\star \in \mathscr{Y}^{\mathscr{X}}$, that minimizes the risk over all possible functions, i.e.,

$$f^* = \underset{f\in\mathscr{Y}^{\mathscr{X}}}{\texttt{arginf}}\Big\{R(f)\Big\} = \underset{f\in\mathscr{Y}^{\mathscr{X}}}{\texttt{arginf}}\Big\{\mathbb{E}[\mathcal{L}(Y, f(\mathbf{x}))]\Big\}. \tag{1.7}$$

Partly due to the fact the universal space $\mathscr{Y}^{\mathscr{X}}$ is infinitely large, and also crucially to the fact that $p_{xy}(\mathbf{x}, y)$ is never known in practice, the ideal universe best $f^\star$ of (1.7) is never known in practice. Instead, a slightly less intangible way to compare functions (learning machines) is to define the risk function within the function class $\mathscr{H}$, namely

$$R_{\mathscr{H}}(f) = \mathbb{E}\Big[\mathcal{L}(Y, f(X)) | f \in \mathscr{H}\Big] = \int_{\mathscr{X}\times\mathscr{Y}} \mathcal{L}(\mathbf{x}, y) p_{xy}(\mathbf{x}, y) d\mathbf{x}dy, \tag{1.8}$$

and seek $f^\star_{\mathscr{H}} \in \mathscr{H}$ such that

$$f^\star_{\mathscr{H}} = \underset{f \in \mathscr{H}}{\texttt{arginf}} \Big\{ R_{\mathscr{H}}(f) \Big\}. \tag{1.9}$$

Note that $f^\star_{\mathscr{H}} \in \mathscr{H}$ is not as universal as $f^\star \in \mathscr{Y}^{\mathscr{X}}$ because the former is restricted to the function space $\mathscr{H}$ while the latter has no restriction, being universal. Throughout this thesis, (1.9) will play a central role, because we will consider different function spaces, and it will be of interest to find out which one performs better. For clarity, given two function spaces $\mathscr{H}_1$ and $\mathscr{H}_2$, we would prefer $\mathscr{H}_1$ over $\mathscr{H}_2$ is the realized best function from $\mathscr{H}_1$ has smaller theoretical risk than the realized best function from $\mathscr{H}_2$. In other words,

$$\texttt{If } R^\star_{\mathscr{H}_1} < R^\star_{\mathscr{H}_2} \qquad \texttt{choose } \mathscr{H}_1, \tag{1.10}$$

where $R^\star_{\mathscr{H}_1} = R_{\mathscr{H}_1}(f^\star_{\mathscr{H}_1})$ and $R^\star_{\mathscr{H}_2} = R_{\mathscr{H}_1}(f^\star_{\mathscr{H}_2})$.

Specifically in this thesis we are interested in two function spaces, namely the function space $\mathscr{H}_{\texttt{DNN}}$ of deep neural networks and the function space $\mathscr{H}_{\texttt{KLM}}$ of kernel learning machines. As stated earlier, our overarching goal in this thesis is find out if

$$R^\star_{\mathscr{H}_{\texttt{DNN}}} < R^\star_{\mathscr{H}_{\texttt{KLM}}}. \tag{1.11}$$

It turns out that checking the inequality in (1.11) as it is constitutes a gigantic theoretical task far beyond the scope of this thesis, partly due to the fact that the theoretical manipulations involved are either not even known yet or are very complex. Rather than seeking to compare the true risks for each of the function spaces, the thesis adopts adopt the comparison of various statistics around the test error computed from the given data $\mathscr{D}_n$. Recall that $R_{\mathscr{H}}(f)$ for a function space $\mathscr{H}$ is the generalization error or true error of f in space $\mathscr{H}$. Stochastic Hold Out is widely used for estimating the generalization error of statistical machine learning models as discussed in Fokoué, 2020. As depicted in Algorithm (1), the given dataset $\mathscr{D}_n$ is randomly split into a training and test repeated, and summaries of the replicas of the test error are used as estimates of the generalization error.

Algorithm 1: Stochastic Hold Out for Generalization

for $s = 1$ *to* S **do**

Generate the $\mathtt{s}^{th}$ random split of $\mathscr{D}_n$ into $\mathscr{D}^{(s)}_{\texttt{tr}}$ and $\mathscr{D}^{(s)}_{\texttt{te}}$, such that and $|\mathscr{D}^{(s)}_{\texttt{te}}| = (1-\tau)|\mathscr{D}_n|$
and $\mathscr{D}_n = \mathscr{D}^{(s)}_{\texttt{tr}} \cup \mathscr{D}^{(s)}_{\texttt{te}}$, and $n = |\mathscr{D}_n| = |\mathscr{D}^{(s)}_{\texttt{tr}}| + |\mathscr{D}^{(s)}_{\texttt{te}}|$

for $m = 1$ *to* M **do**

Build and refine the m^{th} learning machine $\widehat{f}_m^{(\mathscr{D}^{(s)}_{\texttt{tr}})}(\cdot)$ using $\mathscr{D}^{(s)}_{\texttt{tr}}$
Compute predictions $\widehat{f}_m^{(\mathscr{D}^{(s)}_{\texttt{tr}})}(\mathbf{x}_i)$ $\texttt{for } \mathbf{z}_i \in \mathscr{D}^{(s)}_{\texttt{te}}$
Compute the test error for the m^{th} learning machine

$$\widehat{\varepsilon}_{sm} = \widehat{R}_{\texttt{te}}(\hat{f}^{(s)}_m) = \frac{1}{|\mathscr{D}^{(s)}_{\texttt{te}}|} \sum_{i=1}^{n} \mathbb{1}(\mathbf{z}_i \in \mathscr{D}^{(s)}_{\texttt{te}}) \mathcal{L}(\mathrm{y}_i, \widehat{f}_m^{(\mathscr{D}^{(s)}_{\texttt{tr}})}(\mathbf{x}_i))$$

Essentially then, given a dataset $\mathscr{D}_n$ and a collection of potential function spaces $\mathscr{H}_1, \mathscr{H}_2, \cdots, \mathscr{H}_M$, we set a number S of replications (splits), along with a proportion

$\tau \in (0, 1/2)$ of the data $\mathscr{D}_n$ to be allocated to the test set at each split. Upon splitting S times using $\mathscr{D}_n^{(\mathtt{s})} = \mathscr{D}_{\mathtt{tr}}^{(\mathtt{s})} \cup \mathscr{D}_{\mathtt{te}}^{(\mathtt{s})}$, we create for each dataset $\mathscr{D}_n$, the matrix E of realizations of the test error as seen in (1.12).

$$E = \begin{bmatrix} \varepsilon_{11} & \varepsilon_{12} & \cdots & \varepsilon_{1m} & \cdots & \varepsilon_{1M} \\ \varepsilon_{21} & \varepsilon_{22} & \cdots & \varepsilon_{2m} & \cdots & \varepsilon_{2M} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varepsilon_{s1} & \varepsilon_{s2} & \cdots & \varepsilon_{sm} & \cdots & \varepsilon_{sM} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varepsilon_{S1} & \varepsilon_{S2} & \cdots & \varepsilon_{Sm} & \cdots & \varepsilon_{SM} \end{bmatrix} \tag{1.12}$$

For further clarity, it is important to see for the matrix E, that

$$\begin{aligned} \varepsilon_{sm} &= \widehat{R}_{\mathtt{te}}(\hat{f}_m^{(s)}) = \mathtt{te}(\widehat{f}_m^{(\mathscr{D}_{\mathtt{tr}}^{(\mathtt{s})})}) \\ &= \texttt{Error of } \widehat{f}_m^{(\mathscr{D}_{\mathtt{tr}}^{(\mathtt{s})})}(\cdot) \text{ on } \mathscr{D}_{\mathtt{te}}^{(\mathtt{s})}. \end{aligned} \tag{1.13}$$

where

$$\widehat{R}_{\mathtt{te}}(f) = \frac{1}{|\mathscr{D}_{\mathtt{te}}|} \sum_{j=1}^{n} \mathcal{L}(\mathbf{y}_j, f(\mathbf{x}_j)) \mathbb{1}(\mathbf{z}_j \in \mathscr{D}_{\mathtt{te}}), \tag{1.14}$$

and crucially, $\widehat{f}_m^{(\mathscr{D}_{\mathtt{tr}}^{(\mathtt{s})})}$ is the best learning machine from function space $\mathscr{H}_m$, learned using the sth training split $\mathscr{D}_{\mathtt{tr}}^{(\mathtt{s})}$ along with refinement and model selection via V-fold cross validation.

$$\widehat{f}_m^{(\mathscr{D}_{\mathtt{tr}}^{(\mathtt{s})})} := \underset{\widehat{f} \in \mathscr{H}_m}{\mathtt{argmin}} \left\{ \mathtt{CV}(\widehat{f}; \mathscr{D}_{\mathtt{tr}}^{(\mathtt{s})}) \right\} \tag{1.15}$$

It is worth repeating here that ε_{sm} which the realized test error of the mth learning machine on the sth replicate (split) of the data $\mathscr{D}_n$, is the most important ingredient for our overarching goal. In order to perform as thorough and complete a comparison of the estimated generalization errors as possible, we consider several statistical summarises of the values of $\varepsilon_{sm}, s = 1, \cdots, S,\ m = 1, \cdots, M$. As will be seen in Chapter 5, we will consider for each data set, an empirical counterpart of the theoretical generalization errors defined in Equation (1.10), namely, we will define

$$\mathtt{score}(\mathscr{H}_m | \mathscr{D}_n) := \widehat{R}^{\star}_{\mathscr{H}_m} := \text{Empirical counterpart of } R^{\star}_{\mathscr{H}_m} \texttt{ given } \quad \mathscr{D}_n. \tag{1.16}$$

For regression learning tasks, we end up considering the following scores. Since we do consider several different datasets in the spirit of testing the so-called no free lunch theorem (NFLT), it makes sense to reveal the dataset in the expression of ε_{sm}, maybe by writing $\varepsilon_{sm}^{(\mathscr{D}_n)}$.

1. `Average Test Error Criterion`

$$\mathtt{score}(\mathscr{H}_m | \mathscr{D}_n) := \frac{1}{S} \sum_{\mathtt{s}=1}^{S} \varepsilon_{sm}^{(\mathscr{D}_n)}. \tag{1.17}$$

2. `Median Test Error Criterion`

$$\mathtt{score}(\mathscr{H}_m | \mathscr{D}_n) := \underset{s=1,\cdots,S}{\mathtt{median}} \{ \varepsilon_{sm}^{(\mathscr{D}_n)} \}. \tag{1.18}$$

3. `Maximum Test Error Criterion`

$$\texttt{score}(\mathscr{H}_m|\mathscr{D}_n) := \underset{s=1,\cdots,S}{\texttt{max}}\{\varepsilon_{sm}^{(\mathscr{D}_n)}\}. \tag{1.19}$$

4. `Minimum Test Error Criterion`

$$\texttt{score}(\mathscr{H}_m|\mathscr{D}_n) := \underset{s=1,\cdots,S}{\texttt{min}}\{\varepsilon_{sm}^{(\mathscr{D}_n)}\}. \tag{1.20}$$

For any given data set, we will above scores as our criteria, and we will declare as winner the $\mathscr{H}_m$ with the smallest $\texttt{score}(\mathscr{H}_m|\mathscr{D}_n)$.

For classification learning tasks, we will use the same criteria as in regression learning, but this the test accuracy will replace of the test error, namely

$$\pi_{sm}^{(\mathscr{D}_n)} := 1 - \varepsilon_{sm}^{(\mathscr{D}_n)}.$$

Of course, this time, the winner will the $\mathscr{H}_m$ with the highest score.

Equipped with all the above scores and criteria, this thesis ultimately performs an empirical comparison of the generalization performances of kernel learning machines against Deep Neural Networks. We use a wide variety of datasets differing in size, shape, and data types, with the finality of finding out if there is any evidence that Deep Neural Networks are predictively better across all datasets are claimed by some practitioners. For thoroughness and completeness, we include both simulated data and real-world data; we also consider datasets covering lots of different areas, such as the medical sector, the economy, the transportation sector, etc. This thesis does not to delve deep into the methodological and technical subtleties and details of Deep Neural Networks and the corresponding estimation and prediction mechanisms of deep learning, yet we believe that the thorough empirical comparisons of the predictive performances of the machines do provide useful insights into their respective strengths and weaknesses, which we deem of paramount importance to practitioners.

1.3 Organization

The rest of this thesis is organized as follows: Chapter 2 provides a standard description of the architecture of Deep Neural Networks along with some elements of their fundamental characteristics. Chapter 3 touches on the general description of the kernel machines explored in this paper, focusing on Gaussian process-inspired learning machines and support vector machines. Chapter 4 features some theoretical results and relationships mentioned earlier between Deep Neural Networks and kernel machines. Chapter 5 gives a detailed description of the experimental setup of our extensive computational comparison, focusing on the effect of the ratio of the sample size to the dimensionality of the input space, on the one hand, and the sheer diversity of entire data sets on the other. It clearly shows the metrics used in our comparisons of predictive performances. This chapter also presents the detailed computations with the actual comparisons of predictive performances and all the relevant corresponding comments. Chapter 6 presents our discussion and conclusion and the points we intend to explore in our future work on this fascinating and fast-developing theme.

Chapter 2

Neural Networks Learning Machines

A Deep Neural Network (DNN) is an artificial neural network with multiple layers between the input and output layers. It is a highly parameterized machine inspired by the architecture of the human brain. Despite the existence of different kinds of neural networks, they all always have in common the same components, namely: neurons, weights, and activation functions. As we indicated right from the beginning, the neural network learning paradigm has had so many practical and impactful applications in real life that it makes sense for us to provide a refresher of their fundamental building blocks. This chapter aims to provide a standard description of the architecture of neural networks and deep neural networks in particular, with a focus of some of their essential elements. We will not cover the actual learning process used in the mechanics of deep learning because our ultimate treatment in this thesis computational rather than methodological. Our description in this chapter is aimed at highlighting those aspects of Deep Neural Networks that most directly impact the crux of generalization that constitutes the goal of this thesis.

2.1 Functions Space for Deep Neural Networks

As we said before in Chapter 1, supervised learning typically requires the specification of function space $\mathscr{H}$ assumed by the researcher/experimenter as the set from which the best fit from the data is drawn. The overarching goal of this thesis is ultimately a comparison of the generalization performances of deep neural networks against those of kernel learning machines.

2.1.1 Definition of the Function Space $\mathscr{H}_{\text{dnn}}$

According to the comparison criterion defined in Chapter 1, namely (1.10), we really need to define the function space that characterizes Deep Neural Networks (DNN). Using the input space $\mathscr{X}$ and the output space $\mathscr{Y}$, the function space that characterizes a generic DNN is given by (2.1) below:

$$\mathscr{H}_{\text{dnn}} := \left\{ \mathbf{x} \mapsto \psi(\boldsymbol{W}^{(L+1)}\psi(\boldsymbol{W}^{(L)}\psi(\cdots \boldsymbol{W}^{(2)}\psi(\boldsymbol{W}^{(1)}\mathbf{x} + \mathbf{w}_0^{(1)}) + \mathbf{w}_0^{(2)})\cdots) + \mathbf{w}_0^{(L+1)}) \right\}, \tag{2.1}$$

where $\psi(\cdot)$ is the so-called activation function, herein applied vector-wise on the output from the previous layer. These activation functions, especially the ones appearing in the hidden layers, are intended to help capture nonlinearities. The activation

vector $\boldsymbol{a}^{(\ell)} = (1, a_1^{(\ell)}, a_2^{(\ell)}, \cdots, a_m^{(\ell)}, \cdots, a_{d_\ell}^{(\ell)})^\top$ in the ℓ^{th} layer allows the component-wise transformation

$$z_m^{(\ell)} = w_{m,0}^{(\ell-1)} + \sum_{c=1}^{d_{\ell-1}} w_{m,c}{}^{(\ell-1)} a_c^{(\ell-1)}, \tag{2.2}$$

which is simply written in vector form as

$$\boldsymbol{z}^{(\ell)} = \boldsymbol{W}^{(\ell-1)} \boldsymbol{a}^{(\ell-1)} \quad \text{where} \quad a_m^{(\ell)} = h^{(\ell)}(z_m^{(\ell)}). \tag{2.3}$$

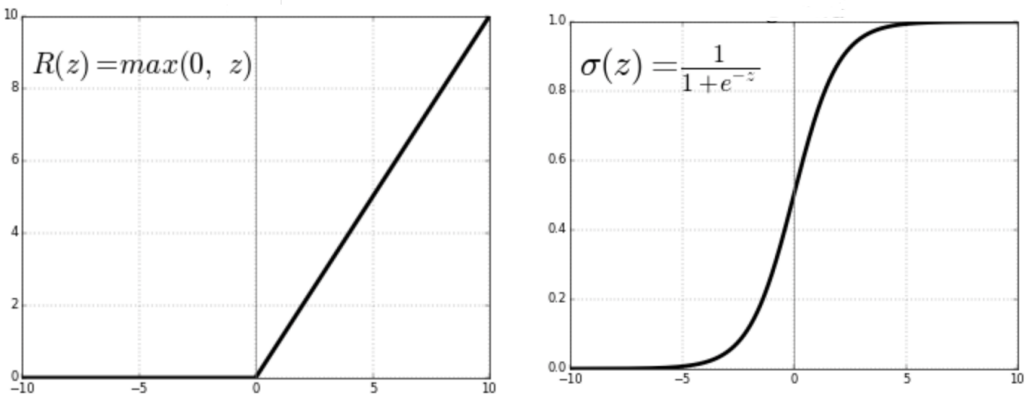
More succinctly, $h^{(\ell)}(z^{(\ell)}) = \psi(\boldsymbol{W}^{(\ell)} \boldsymbol{z}^{(\ell)} + \mathbf{w}_0^{(\ell)})$; These latent quantities play in central role in the neural networks learning paradigm.

2.1.2 Activation Functions

The Activation Function is a critical part of Deep Neural Networks. It defines how the inputs weighted sum is transformed into an output from a node or nodes in a layer of the network. According to the type of prediction problems, one could carefully choose a suitable activation function.Numerous activation functions already exist. One of the newer activation functions called the ReLU is defined by:(Nair and Hinton, 2010)

$$\psi(u) = \max(0, u) = (u)_+$$

It is one of the prevalent activation functions being used for the hidden layers. It is simple but very efficient when compared to its peer activation functions (Figure 2.1a). One advantages of ReLu is that its less susceptible to vanish gradients which prevent Deep Neural Network models from being trained. However, it does have some problems such as "dead" units. As stated earlier, there are many kinds of activation functions, here are a few examples:



(A) ReLu (B) Sigmoid

FIGURE 2.1: Activation function

- The logistic sigmoid:

$$\psi(u) = \frac{1}{1 + e^{-u}}$$

which is one of the most commonly used activation function(Figure 2.1b);

- The *softmax*:

$$\psi(\mathbf{u}) = \texttt{softmax}(\mathbf{u}) = \frac{e^{\mathrm{u}_g}}{\sum_{j=1}^{G} e^{\mathrm{u}_j}}$$

  which adapted for multi-categorical tasks to *softmax*;

- Radial basis activation function:

$$\psi(\mathrm{u}) = \exp\left(-\tau \mathrm{u}^2\right)$$

- Linear activation function:

$$\psi(u) = \boldsymbol{\alpha} + u^T \boldsymbol{\beta}$$

The other aspect of neural network of paramount importance is the parameter space as seen in the subsequent sections of this thesis.

2.2 Parameter space for Deep Neural Networks

It turns out that one of the most challenging bottlenecks with deep neural networks stems from the sheer large numbers of weights to be learned/estimated. In fact, in (2.1), the collection $\boldsymbol{\theta} := \{\boldsymbol{W}^{(\ell)} : \ \ell = 1, \cdots, L+1\}$ of all the weight matrices , forms the set of parameters for the corresponding DNN model.

$$\boldsymbol{W}^{(\ell)} = \begin{bmatrix} \mathrm{w}_{1,0}^{(\ell)} & \mathrm{w}_{1,1}^{(\ell)} & \mathrm{w}_{1,2}^{(\ell)} & \cdots & \mathrm{w}_{1,c}^{(\ell)} & \cdots & \mathrm{w}_{1,d_{\ell-1}}^{(\ell)} \\ \mathrm{w}_{2,0}^{(\ell)} & \mathrm{w}_{2,1}^{(\ell)} & \mathrm{w}_{2,2}^{(\ell)} & \cdots & \mathrm{w}_{2,c}^{(\ell)} & \cdots & \mathrm{w}_{2,d_{\ell-1}}^{(\ell)} \\ \vdots & \vdots & \vdots & \cdots & \ddots & \cdots & \vdots \\ \mathrm{w}_{m,0}^{(\ell)} & \mathrm{w}_{m,1}^{(\ell)} & \mathrm{w}_{m,2}^{(\ell)} & \cdots & \mathrm{w}_{m,c}^{(\ell)} & \cdots & \mathrm{w}_{m,d_{\ell-1}}^{(\ell)} \\ \vdots & \vdots & \vdots & \cdots & \ddots & \cdots & \vdots \\ \mathrm{w}_{d_\ell,0}^{(\ell)} & \mathrm{w}_{d_\ell,1}^{(\ell)} & \mathrm{w}_{d_\ell,2}^{(\ell)} & \cdots & \mathrm{w}_{d_\ell,c}^{(\ell)} & \cdots & \mathrm{w}_{d_\ell,d_{\ell-1}}^{(\ell)} \end{bmatrix}. \tag{2.4}$$

Figure 2.2 shows a typical structure of a deep neural networks machine. There are three layers: the input layer, hidden layers, and the output layer. However, there could be more layers within the hidden layer and each hidden layer can also contain many neurons.

2.2.1 Complexity of Deep Neural Networks

One of the most immediate remarks about Deep Neural Networks has do with their complexity or more simply put their size or number of parameters needed to be estimated from the data. Clearly, given that each weight matrix $\boldsymbol{W}^{(\ell)}$ is potentially quite large even for modest tasks, it is often the case that $\boldsymbol{\theta}$ has a typically very large number of entries entries even for very moderate values of L since each $\boldsymbol{W}^{(\ell)}$ is itself already a large $d_\ell \times (d_{\ell-1}+1)$ matrix. If $dim(\mathscr{X}) = q$, and $dim(\mathscr{Y}) = r$, and $dim(\mathscr{Z}_\ell) = d_\ell$, then for a DNN with L hidden layers, $\boldsymbol{\theta}$ will have $p = q + r + \sum_{\ell=1}^{L} d_\ell \times (d_\ell + 1)$ entries, at least a priori. Now, given a data set $\mathscr{D}_n = \{(\mathbf{x}_i, y_i) \overset{iid}{\sim} p_{xy}(\mathbf{x}, y),\ \mathbf{x}_i \in \mathscr{X},\ y_i \in \mathscr{Y},\ i = 1, \cdots, n\}$, the only hope of

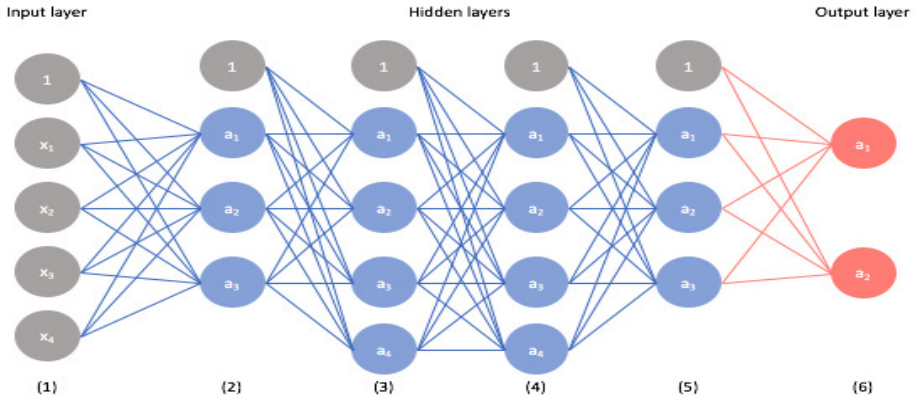


FIGURE 2.2: Pictorial Representation of a generic Deep Neural Network. This example shows a DNN with an input layer, 4 hidden layers, and an output layer.

learning a unique DNN rests on having $n > p$, ie more observations that parameters needed to be estimated. It is crucial to note that for many tasks of great interest to practitioners, one will typically have situations were $n \lll p$, putting a dent on the potential for DNN to be the right method for such tasks.

The number of Hidden Layers and Neurons determines the size of the Neural Networks (NN) model. For instance, the number of nodes determines the parameters for a single hidden layer Neural Networks machine. The number of nodes is also treated as a parameter. Because overfitting is one of the problems while building an NN machine, we need to wisely control the number of nodes. The number of layers is another parameter that, along with the number of neurons in each layer, will control the complexity of a NN machine. Instead of weight regularization, tuning these parameters to control the complexity would be more efficient.

2.3 Deep Learning: Training Neural Networks

2.3.1 Empirical Risk Minimization and Deep Learning

As we can see so far, a Neural Networks model is very complex. $f(\mathbf{x}; \boldsymbol{W}) \in \mathscr{H}k$ is a hierarchical function of the vector $\mathbf{x}$ and the weight collection $\boldsymbol{W}$. For a given regression training data set $\mathscr{D}_n = \{(\mathbf{x}_i, y_i) \overset{iid}{\sim} p_{xy}(\mathbf{x}, y),\ \mathbf{x}_i \in \mathscr{X},\ y_i \in \mathscr{Y},\ i = 1, \cdots, n\}$ and the loss function $\mathcal{L}(y, f(\mathbf{x}))$, we want to solve

$$\underset{W}{\texttt{minimize}} \left\{ \frac{1}{n} \mathcal{L}[y_i, f(\mathbf{x}_i, \boldsymbol{W}))] + \alpha J(\boldsymbol{W}) \right\} \tag{2.5}$$

α is a tuning parameter, and $J(\boldsymbol{W})$ is a regularization term.

2.3.2 Dropout

Dropout in Neural Networks refers to dropping some neurons in layers. It is a form of regularization and always used when learning a Neural Networks model. One can apply different dropout rates to different layers. Dropout always works better for DNN with the deeper and denser layers. The dropout rate is between 0 to 1; 0

means no dropout, 1 means no output. Dropout has been widely used to prevent the overfitting of Deep Neural Networks with many parameters. Goodfellow, Bengio, and Courville, 2016

> dropout is more effective than other standard computationally inexpensive regularizers, such as weight decay, filter norm constraints and sparse activity regularization. Dropout may also be combined with other forms of regularization to yield a further improvement.

$$\hat{\mathbf{w}}_i = \begin{cases} \mathbf{w}_i & \text{if } p \neq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2.6}$$

where p is the probability to keep a row in the weight matrix, $\mathbf{w}_i$ is the real row in the weight matrix, $\hat{\mathbf{w}}_i$ is the weight matrix after dropout.
Even though dropout appears to work quite well for many DNN architecture, Murray and Fokoué, 2021 just recently established that dropout fails to regularize for nonparametric learners. Partly in response to the challenges paused by the complexity of deep neural networks, especially in training, different types of neural network architectures have been invented including reservoir computing learning machines, of which echo state networks are a well known example. Wu, Fokoué, and Kudithipudi, 2018 provide a quick review of the fundamentals of echo state networks as well as a characterization of the statistical and probabilistic properties of the weights of the hidden layers.

There are some other parameters in the Neural Networks machines. For instance, the choice of regularization is typical of a l_1 and l_2 mixture problem, and each of them requires a tuning parameter. The l_2 reduces problems with collinearity, and the l_1 can ignore irrelevant features. Both slow the rate of overfitting, especially with over-parametrized networks. Another parameter is "Early Stop", but we are less concerned about this parameter since the regularization is tuned adaptively to avoid overfitting in a NN model.

Chapter 3

Kernel Learning Machines

3.1 An Introduction to Kernel Methods

3.1.1 Generalities on kernel methods

In the late 1990s, kernel methods and kernel learning machines gained tremendous popularity thanks to Support Vector Machines, Gaussian Process and Radial basis function networks and relevance vector machines a bit later. Kernels and kernel learning machines became into so popular that researchers sought to so-called "kernelized" as many existing methods as possible, leading to powerful machines as kernel PCA, kernel kMeans, kernel CCA, kernel regression just to name a few. In fact, kernels have now touched both supervised and unsupervised learning very deeply, extending even to reinforcement learning and beyond. Kernels have served as the backbone of many data science methods since their very inception Fokoué and Brimkov, 2018. It is worth noting that kernel Learning has also be applied to ranking Regression Guimaraes Olinto and Fokoué, 2018. The key lies in the fact that kernel methods consists internally of a transformation of the data into some feature space that usually has a relatively larger dimension. Even though the dimension gets larger, Baudat and Anouar, 2001 shows that the capacity of a generalization depends on the geometrical characteristics of the training data, and not on the dimension of the input space. Interestingly and somewhat crucially, the direct use of a kernel function reduces the complexity of finding the mapping function (Karatzoglou et al., 2004; Clarke, Fokoue, and Zhang, 2009). This is known as the kernel trick. Kernel function allow the implicit computation of the feature space calculations with the function defined in the input space.

Kernels are used such that a point in the dataset will affect the nearby points more than it affects the further away points. This is the reason why kernels in the sense that we use them in this thesis are essentially *measures of similarities*.

For the purposes of the overarching goal of comparing predictive performances of learning machines, it turns out that with suitable geometrical characteristics, the generalization error could get smaller with the suitable kernel even though the feature space has a large dimension. What really do we mean by a kernel? It is important to clarify this because the word kernel means different things even in different areas of the same field of Mathematics.

3.1.2 Definition of a kernel

Throughout this thesis, a kernel $\mathcal{K}$, is a bivariate function defined on the input space. For our purposes, it measures the similarity between two given elements from $\mathscr{X}$

$$\begin{aligned} \mathcal{K}: \quad & \mathscr{X} \times \mathscr{X} \longrightarrow \mathbb{R}_+ \\ & (\mathbf{x}_l, \mathbf{x}_m) \longmapsto \mathcal{K}(\mathbf{x}_l, \mathbf{x}_m) \end{aligned} \tag{3.1}$$

Here are some commonly used functions,

- Polynomial kernel with degree d:

$$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = (\boldsymbol{\beta}\mathbf{x}_i^\top \mathbf{x}_j + c)^d \tag{3.2}$$

  where d is the degree of polynomial, b is the scale parameter and c is the offset parameter, known as performing very well on image processing tasks;

- Linear kernel (vanilla):

$$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j \tag{3.3}$$

  which is the simplest among all the kernel function, and it works on the high dimensional sparse data;

- Gaussian radial basis function:

$$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma(\mathbf{x}_i - \mathbf{x}_j)^2} = e^{-\gamma(\mathbf{x}_i - \mathbf{x}_j)^\top (\mathbf{x}_i - \mathbf{x}_j)} \tag{3.4}$$

  this is the most used kernel when there is no prior knowledge on the data, and γ is the bandwidth, by changing γ to help with solving a complex problem model;

- Laplace radial basis kernel:

$$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma||\mathbf{x}_i - \mathbf{x}_j||} \tag{3.5}$$

  γ is the bandwidth, and it used in the same context as the Gaussian kernel. There are some other kernel functions. (Karatzoglou et al., 2004 Clarke, Fokoue, and Zhang, 2009)

Once a kernel $\mathcal{K}$ is chosen for the task of interest, the so-called Gram matrix $\boldsymbol{K} = (\mathcal{K}(\mathbf{x}_l, \mathbf{x}_m)), l.m = 1, \cdots, n$ is formed and constitutes the most important object from then on. It is easy to see that the Gram matrix $\boldsymbol{K}$ plays a role similar to the design matrix or data matrix $\boldsymbol{X}$ in linear models.

$$\boldsymbol{K} := \begin{bmatrix} \mathcal{K}(\mathbf{x}_1, \mathbf{x}_1) & \mathcal{K}(\mathbf{x}_1, \mathbf{x}_2) & \cdots & \mathcal{K}(\mathbf{x}_1, \mathbf{x}_n) \\ \mathcal{K}(\mathbf{x}_2, \mathbf{x}_1) & \mathcal{K}(\mathbf{x}_2, \mathbf{x}_2) & \cdots & \mathcal{K}(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \cdots & \vdots \\ \mathcal{K}(\mathbf{x}_i, \mathbf{x}_1) & \mathcal{K}(\mathbf{x}_i, \mathbf{x}_2) & \cdots & \mathcal{K}(\mathbf{x}_i, \mathbf{x}_n) \\ \vdots & \vdots & \cdots & \vdots \\ \mathcal{K}(\mathbf{x}_n, \mathbf{x}_1) & \mathcal{K}(\mathbf{x}_n, \mathbf{x}_2) & \cdots & \mathcal{K}(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \tag{3.6}$$

Some learning machines like support vector machines do indeed require the kernel to be positive definite in order to guarantee many convergence and stability properties of the learning process.

3.1.3 Function Space for Kernel Machines $\mathscr{H}_{\texttt{klm}}$

Just like we did before with neural networks in Chapter 2, it is important to specify what the function space looks like when one is dealing kernel learning machines. Given a dataset $\mathscr{D}_n = \{(\mathbf{x}_i, y_i) \overset{iid}{\sim} p_{xy}(\mathbf{x}, y),\ \mathbf{x}_i \in \mathscr{X},\ y_i \in \mathscr{Y},\ i = 1, \cdots, n\}$ and a suitable chosen kernel $\mathcal{K}$, the corresponding function space is given by

$$\mathscr{H}_{\texttt{klm}} := \mathscr{H}_{\mathcal{K}} := \left\{ \mathbf{x} \mapsto \varphi\left(\sum_{j=1}^{n} \mathrm{w}_j \mathcal{K}(\mathbf{x}, \mathbf{x}_j) + \mathrm{w}_0 \right),\ \mathrm{w}_0 \in \mathbb{R},\ \mathrm{w}_j \in \mathbb{R},\ j \in [n] \right\}, \quad (3.7)$$

$\varphi(\mathrm{v}) = \mathrm{v}$ for regression or $\varphi(\mathrm{v}) = \texttt{sign}(\mathrm{v})$ for binary classification with labels $\{-1, +1\}$. From a pure representation point of view, regression and classification are not structurally different when it comes to kernel learning machines. Of course, the actual mechanics of learning can differ quite substantially in some cases, especially because the change of loss functions can result in substantially changes in the learning process. The actual development of kernel learning machines is beyond the scope of this thesis. We will simply evoke and use all the kernel learning machines of interest in the final forms. It is worth noting immediately that for any $f \in \mathscr{H}_{\texttt{klm}}$, we have

$$f(\mathbf{x}) = \varphi\left(\sum_{j=1}^{n} \mathrm{w}_j \mathcal{K}(\mathbf{x}, \mathbf{x}_j) + \mathrm{w}_0 \right), \quad (3.8)$$

so that any kernel learning machine in the sense intended in this work has at more $n + 1$ weights playing the role of "parameters". All the kernel learning machines used throughout this research of the form depicted in Equation (3.8), regardless of whether the task is regression or binary classification. Multiclass classification is handled slightly differently but with exactly the same main foundational form.

3.2 Kernel learning machines

Kernel methods provide very powerful tools in the portfolio of statistical machine learning techniques. As a matter of fact, lots of algorithms can operate and indeed have operated with kernels as we mentioned earlier. It is well-known that the very popular support vector machines (SVM) are kernel learning machines. However, for our purposes, we will also Gaussian processes that are powerful kernel learning machines in their own right. Both of these will ultimately compared with Deep Neural Networks in Chapter 4 and Chapter 5.

3.2.1 Support Vector Machines

Support Vector Machines (SVM) are popular machine learning tools for both classification and regression. They were invented by Vladimir Vapnik and Alexis Chervonenkis, and later developed and expanded by several other researchers around the world, after they were found to be extremely powerful on several difficult problems, especially in high dimensional tasks (Cortes and Vapnik, 1995; Drucker et al., 1997; Scholkopf and Smola, 2018).

The main differences between Support Vector Regression and Support Vector Classification are the noise model and/or loss function but the paradigm of maximization of a margin is still the same. Here are the typical expressions of a Support Vector binary Classification model and a Support Vector regression model.
Being kernel learning machines, support vector machines are of the form of Equation (3.8), with only difference being in the way the weights are learned.

Support Vector Binary Classification: For a binary classification problem, Support Vector Machine with kernel could be expressed as following:

$$\widehat{f}_{\texttt{SVM}}(\mathbf{x}) = \texttt{sign}\left(\sum_{i=1}^{n} \widehat{\boldsymbol{\alpha}}_i \mathbf{y}_i \mathcal{K}(\mathbf{x}_i, \mathbf{x}) + \widehat{\mathrm{w}}_0\right) \tag{3.9}$$

$\widehat{\boldsymbol{\alpha}}_i$, $i = 1, \cdots, n$ are the Lagrange multipliers of used on the constrained optimization task that defines the celebrated Support Vector Machine. The $\widehat{\boldsymbol{\alpha}}_i$, $i = 1, \cdots, n$ are estimated via quadratic programming. Those vectors $\mathbf{x}_i$ for which $\widehat{\boldsymbol{\alpha}}_i \neq 0$ are correspond to the support vectors that gave the name to the machine.

$\widehat{f}_{\texttt{SVM}}(\mathbf{x}) \in \{-1, +1\}$ is the kernelized binary classifier's predicted label for the input $\mathbf{x}$ whose true label $\mathbf{y}$ is being predicted;

Support Vector Regression Learning: The support Vector Regression Learning machine is of the following form:

$$\widehat{f}_{\texttt{SVM}}(\mathbf{x}) = \sum_{i=1}^{n} (\widehat{\boldsymbol{\alpha}}_i - \widehat{\boldsymbol{\alpha}}_i^{\star}) \mathcal{K}(\mathbf{x}_i, \mathbf{x}) + \widehat{\mathrm{w}}_0 \tag{3.10}$$

where $\widehat{\boldsymbol{\alpha}}_i, \widehat{\boldsymbol{\alpha}}_i^{\star}$, $i = 1, \cdots, n$ are once again the Lagrange multipliers just like before and are estimated via quadratic programming.

3.2.2 Gaussian Process

A Gaussian Process (GP) is a powerful statistical learning machine (Rasmussen and Williams, 2005), that builds for several convenient properties of the multivariate normal (Gaussian) distribution. Gaussian Process models are non-parametric probabilistic models. Gaussian Process models can be naturally expressed in the Bayesian inference framework by using kernel functions or other covariance functions. Gaussian Processes are a powerful technique for modeling and making prediction on all kinds of data. They are very flexible and can be used to model many different patterns. They need fewer assumptions about the model compared to other machines. Just like with Support Vector Machines, it turns out that Gaussian Processes can be conveniently expressed using the form of Equation (3.8). Specifically, for Gaussian Process we have:

$$\widehat{f}_{\texttt{gpr}}(\mathbf{x}) = \varphi\left(\sum_{j=1}^{n} \widehat{\mathrm{w}}_j \mathcal{K}(\mathbf{x}_j, \mathbf{x}) + \widehat{\mathrm{w}}_0\right). \tag{3.11}$$

For regression under the Gaussian noise model with homoscedastic variance σ^2, $\varphi(\cdot) = \texttt{id}(\cdot)$, and the weights are given by

$$\widehat{\mathrm{w}}_j = [(\boldsymbol{K} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{Y}]_i \tag{3.12}$$

where $\boldsymbol{Y} = (y_1, y_2, \cdots, y_n)^{\top}$ is the n-dimensional vector of all the response values.

Throughout Chapter 4 and Chapter 5, we will repeated make references both these kernel learning machines for specific kernels.

Chapter 4

Conceptual, Methodological and Theoretical Comparisons

As stated several times up to this point, the motivating theme for the thesis is the exploration of similarities and dissimilarities between deep neural networks and kernel learning machines, and hopefully find out if deep neural networks just might be holy grail of statistical machine learning and artificial intelligent. It is important to mention that along with Neural Networks, other paradigms have continued to arise, many of which have consistently challenged the claims of superiority attributed to Neural Networks.

Through Park and Sandberg, 1993, Radial Basis Function Networks are formulated as Neural Networks with one single hidden layer and later shown to be kernel learning machines, with kernels of the specific type known as radial basis function kernels. It's important to note that many other kernel learning machines exist apart from radial basis functions networks, developed from paradigms entirely foreign in principle to Neural Networks, and many of those kernel learning machines have proven to be excellent learning machines built on very solid and unshakable foundations. While the single hidden layer Neural Network enjoys the perch of the universal approximation theorem as its supporting backbone and foundation, it is important to note that kernel learning machines are supported by the equally powerful result known as the Representer Theorem Craven and Wahba, 1979, Wahba, 1990, G., 1998, Wahba, 2000.

Among other things, we will revisit a long studied relationship between single hidden layer Neural Networks and radial basis function networks, especially the similarity of the form of the estimation function. We will also touch on the similarity that arises via the celebrated Universal Approximation Theorem which appears for both radial basis function networks and Single Hidden Layer Neural Networks in almost indistinguishable manner.

We will then explore the appearance of Deep Neural Networks being far more complex and supposedly more powerful predictively and establish that some newer similarities via the so-called neural tangent kernel (NTK) but also some sharp differences, especially computational ones, stemming from the sheer complexity of deep neural network architectures.

4.1 Shallow Neural Networks Comparisons

It is interesting and somewhat profoundly thought-provoking to note that most of the earlier successes of Neural Networks all came thanks to the one single hidden layer incarnation, with the so-called MultiLayer Perceptron (MLP), a single hidden layer feedforward Neural Network leading the charge. If we denote by $\mathscr{H}_{\texttt{1HLNN}}$, the function space of all one single hidden layer Neural Networks, then $\forall f \in \mathscr{H}_{\texttt{1HLNN}}$, we must have $\forall \mathbf{x} \in \mathscr{X} \in \mathbb{R}^d$,

$$f(\mathbf{x}) = \varphi\Big(\boldsymbol{W}^{(2)}\psi\Big(\boldsymbol{W}^{(1)}\mathbf{x} + \mathbf{w}_0^{(1)}\Big) + \mathbf{w}_0^{(2)}\Big). \tag{4.1}$$

In fact, it was around this single hidden layer ($L = 1$) Neural Network that some of the most impressive theoretical foundations of Neural Networks were set and firmly established. Indeed along with all the successes in a wide variety of applications, various versions and incarnations of the so-called *universal approximation theorem* sprang up, all establishing single hidden layer (1HLNN) Neural Networks as formidable learning machines capable of learning any smooth function whether in classification or in regression.

Barron et al., 2008 opened up to the statistics community the desirable functional analysis properties of Neural Networks as a means of universal approximation of functions. In fact, Barron, 1993 even provides an earlier foray in connection with insights into the power of Neural Networks, interestingly based on just a single hidden layer. Arguably one of the earliest if not the absolute earliest account on the universal approximation power of single hidden layer Neural Networks is provided by Cybenko, 1988 and Cybenko, 1989a. Kůrková, 2002 revisits the Universal Approximation Theorem featuring once again the Single hidden Layer multilayer perceptron Neural Networks, further strengthening the perception and acceptance of single hidden layer Neural Networks as formidable learning machines. Interestingly, similarities have long been established between single hidden layer Neural Networks and some kernel learning machines, which begs the question as to whether it even makes sense to fuss about the superiority of one of the learning paradigms over the other.

This work will explore as thoroughly as possible all the similarities and differences between Deep Neural Networks and kernel learning machines. In their heyday, kernel learning machines, spearheaded by radial basis functions networks, also made similar claims of being the holy grail in statistical machine learning and artificial intelligence. Which of the two is better then? Can that question even be answered definitively? In the continuum ranging from rigid parametric models to nonparametric to the so-called semi-parametric models, it might be interesting to revisit the late 1990s theorem that establishes Gaussian Processes as the limit of a Neural Network with an infinite number of nodes in the single hidden layer. Since Gaussian process learning machines are pure nonparametric models this connection may shed some light. What is the function space/class complexity of these good candidate models? For Radial Basis Function Networks, the learning machine is of the form given in Equation (4.1)

$$f(\mathbf{x}) = \varphi\left(\sum_{m=1}^{M} \mathrm{w}_m \psi(\|\mathbf{x} - \mathbf{c}_m\|)\right), \tag{4.2}$$

where $\varphi(\mathrm{v}) = \mathrm{v}$ for regression or $\varphi(\mathrm{v}) = \texttt{sign}(\mathrm{v})$ for binary classification with labels $\{-1, +1\}$. Interestingly, and somewhat crucially, the basis function $\psi(\cdot)$ admits a reformulation in terms of a suitably chosen kernel $\mathcal{K}(\cdot, \cdot)$, namely

$$\psi(\|\mathbf{x} - \mathbf{c}_m\|) = \mathcal{K}(\mathbf{x}, \mathbf{c}_m) = \exp\left(-\frac{1}{2\tau^2}\|\mathbf{x} - \mathbf{c}_m\|_2^2\right) \tag{4.3}$$

The corresponding radial basis function network is then expressed as

$$f(\mathbf{x}) = \psi\left(\sum_{j=1}^{n} \mathrm{w}_j \mathcal{K}(\mathbf{x}, \mathbf{x}_j) + \mathrm{w}_0\right). \tag{4.4}$$

Note that the summation in (4.4) has n terms and uses the $\mathbf{x}_j$'s, whereas the one in uses M terms with $\mathbf{c}_m$ learned from the data. The form however is the same, which makes our point of similarity of two learning machines.

Through a formulation like (4.3), the Radial basis function network which is cast in the context of neural networks finds itself being a bona fide member of the family of kernel learning machines. Note also that Equation (4.1) is similar to Equation (4.1). Interestingly, the similarities do not stop here. In fact, many of the early theoretical results on the universality of neural networks could interchangeably be expressed through radial basis function networks or through the kernel learning machines equivalent, or counterparts.

Theorem 1 *Cybenko, 1989b Let $f : \mathscr{X} \longrightarrow \mathscr{Y}$ be a function of interest to be estimated. Assume that for all $\mathbf{x} \in \mathscr{X}$, $\mathbb{E}[f(\mathbf{x})] < \infty$, and further assume that we have a function $\psi : \mathbb{R} \longrightarrow \mathbb{R}$ that is continuous with $\lim_{\mathbf{x}\to+\infty} \psi(\mathbf{x}) \to 1$ and $\lim_{\mathbf{x}\to-\infty} \psi(\mathbf{x}) \to 0$. Then, for any $\varepsilon > 0$, there exists $n = n(\varepsilon)$, such that*

$$\inf_{\{(a_i, b_i, \mathbf{w}_i)\}} \mathbb{E}\left\{\left|f(\mathbf{x}) - \frac{1}{n}\sum_{i=1}^{n} a_i \psi(\langle \mathbf{w}_i, \mathbf{x}\rangle + b_i)\right|\right\} \leq \varepsilon. \tag{4.5}$$

Several other researchers like Hornik, Stinchcombe, and White, 1989 and Hornik, 1991, have contributed theoretical results of the same type around the approximation capabilities of multilayer feedforward networks, typically featuring the single hidden layer architecture. At this point, it might appear (and rightly so) more interesting to compare single hidden layer Neural Networks to kernel learning machines, thanks in part to intrinsic similarities but also because both paradigms are supported by strong theoretical foundations. Such comparisons will be carried out computationally later. Two natural questions arise in the presence of such an emphatic result as the universal approximation power of single hidden layer neural networks namely:

(a) How can anyone justify the need/importance to study or develop any other learning paradigm if one exists that is a kind of panacea?

(b) Even within the Neural Networks paradigm, why should anyone use more than one hidden layer when one hidden layer has all the approximation power?

4.2 Deep Neural Networks Comparisons

Owing to the rise to prominence of Deep Neural Networks i.e. Neural Networks with more than one hidden layer, it is important to extend our comparison to Deep Neural Networks. Hence our overarching question, namely: *Are Deep Neural Networks (DNN) really predictively better than kernel learning machines (KLM) across the board?* This thought-provoking question, inspired by the surge in interest around Deep Neural Networks, has led many researchers to seriously investigate the predictive performances of Deep Neural Networks relative to other learning paradigms. Specifically, in recent months a number of authors have contributed several accounts (most of them of a computational nature) of the predictive comparisons between Deep Neural Networks and kernel learning machines.

Neal, 1996 provides the seminal account through which Gaussian processes are linked to single hidden layer Neural Networks with an infinite number of nodes in the hidden layer. He specifically established that "when the number of nodes in the single hidden layer of a Neural Networks is allowed to grow to infinity, then one can use a Gaussian process prior over such an infinite network and derive Gaussian process learning machine as a regularized Neural Network. Considering the fact that Gaussian process learning machines are kernel learning machines this result is quite arguably the first and most foundational foray into the now prevalent and widespread effort aimed at comprehending Deep Neural Networks via Gaussian processes. Amongst others Belkin, Ma, and Mandal, 2018 and He et al., 2020 are noteworthy. Belkin, Ma, and Mandal, 2018 declares that *to understand deep learning we need to understand kernel learning*. Taking a computational approach, He et al., 2020 provide a comparison that is very similar in structure and in spirit to ours.

Their very title *Deep neural networks and kernel regression achieve comparable accuracies for functional connectivity prediction of behavior and demographics*, presents the very same conclusion we end up arriving at, namely that the two learning paradigms are very similar in terms of predictive performances.

More recent papers like Lee et al., 2019 centered around Wide Neural Networks and G. Matthews et al., 2018 discussing Gaussian Process Behavior in Wide Deep Neural Networks, can be rightly viewed as extensions of Neal, 1996 this time adapted to Deep Neural Network architectures. Authors like Jacot, Gabriel, and Hongler, 2020 are actively continuing the burgeoning work around the so-called Neural Tangent Kernel (NTK) used as one of the most promising way of establishing the link between Deep Neural Networks and kernel learning machines, with the hope of establishing with DNN the kind of firm similarity and analogy enjoyed by single hidden layer neural networks. Very early on, Cho and Saul, 2009 proposed *Kernel Methods for Deep Learning*, establishing somehow a strong connection between the two paradigms. Both Jacot, Gabriel, and Hongler, 2018 and Jacot, Gabriel, and Hongler, 2020 introduce the Neural Tangent Kernel which provides one of the strongest similarity between deep neural networks and kernel learning machines. This work is truly revolutionary and transformative as it established the strongest connection yet between the two paradigms. Lee et al., 2019 proposes the exploration of *Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent*, following Lee et al., 2018 who formulated *Deep Neural Networks as Gaussian Processes*. Even more recently, Zhang et al., 2021 emphasizes that *Understanding Deep Learning (Still) Requires Rethinking Generalization*. From a theoretical and methodological perspective,

it appears that deep neural networks need kernel learning machines more than the other way around, since we see from many authors that the formulation of DNN via kernels is the pathway towards understanding DNN. In that sense, one may declare the upper hand to kernel learning machines, although the equivalence of formulation makes it hard to give the upper hand to any of the two paradigms.

Maybe the key lies with the kernel itself, in the sense that while the neural tangent kernel might be a suitable device/mechanism for formulating a DNN as a kernel learning machines, it might not be straightforward to compute it. Therefore, one could see a methodological advantage in simply adopting traditional shallow kernels rather than using the neural tangent kernel. Of course since choosing the suitable kernel is akin to model selection which in turn has to do with the typically unknown geometry of the pattern underlying the data, once could conceive of situations where the neural tangent kernel is the optimal choice, albeit with the challenges of it computation.

Finally, it is crucial to mention that classical kernel learning machines commonly used typically require the estimation of at most $n+1$ weights, whereas the number of weights needed for deep neural networks can quickly grow to extremely large numbers. Each of the weight matrices $\boldsymbol{W}^{(\ell)}$ is itself already a large $d_\ell \times (d_{\ell-1}+1)$ matrix. If $dim(\mathscr{X}) = q$, and $dim(\mathscr{Y}) = r$, and $dim(\mathscr{Z}_\ell) = d_\ell$, then for a DNN with L hidden layers, $\boldsymbol{\theta}$ will have $p = q + r + \sum_{\ell=1}^{L} d_\ell \times (d_\ell + 1)$ entries, at least a priori. This makes a generic DNN far more complex than a classical kernel learning machine. We will see in Chapter 5 that this complexity often hurts DNN when there is not enough data to learn all the weights consistently.

Chapter 5

Computational Explorations and Demonstrations

Experimentation is the lifeblood of machine learning research. A significant amount of effort and resources are invested in evaluating the usefulness of algorithms, finding the optimal approach for applications, or just gaining some overall insight (for example: the effect of a parameter). In this research, we intend to adopt an empirical and experimental approach to investigate the predictive strengths of DNN and compare it to Kernel Methods. We will use simulated and real-life data and make them qualitatively and quantitatively different to explore all the paradigms' strengths and weaknesses fully.

This research aims to create a objective comparison between Deep Neural Networks and Kernel learning machines. The fairness is based on variable selection techniques, tuning of parameters, choice of loss functions, and more. We evaluate the machine learning models predictive performances on different types of machine learning tasks. The challenging component is that due to the complex structure and the numerous parameters of Deep Neural Networks, tuning the parameters is very time-consuming and requires plenty of computational power. RIT computer labs and online web servers were able to help solve this problem.

5.1 Tools for Computational Explorations

As we mentioned earlier, we use simulated and real-world dataset in this thesis. For most of the data, we randomly choose 70% of the data as training data and the remaining 30% for the model performance and model evaluation. For each data set, we run 50 replications. To make sure it is objective to all the machines, we normalize the data before we run the methods. We compare the test error among five machines: Gaussian Process, Support vector machine with the linear kernel; Support vector machine with the polynomial kernel; Support vector machine with radial basis function kernel, and Deep Neural Networks.In the rest of this chapter, we will show the details of the data sets, experiments, and results.

Cross Validation: As we indicated right from Chapter 1, all the learning machines considered are tuned and selected via cross validation. Cross validation is a well-known and widely used method for tuning the hyperparameters in statistical learning and data mining. It divides the data into V chunks, and each chunk has almost equal portions, such that $\left\{ \mathscr{D}_n = \sum_{v=1}^{V} \mathscr{D}_v, \text{ for } i = 1, \cdots, V \right\}$, hold out the portion I and fit the model from the rest of the data, then use the fitted model to predict the holdout samples, and the average the measure of predictive performance

over the V different fits to get the cross validation score which is given by

$$CV(\hat{f}) = \frac{1}{V}\sum_{v=1}^{V}\hat{\xi_v} \tag{5.1}$$

where

$$\xi_v = \frac{1}{|\mathscr{D}_v|}\sum_{i=1}^{n} 1(\mathbf{z}_i \in \mathscr{D}_v) L(\mathbf{y}_i, \hat{f}^{(-\mathscr{D}_v)}(\mathbf{x}_i))$$

Algorithm 2: V-fold cross validation

Input: Training data $\mathscr{D}_n = \{\mathbf{z}_i = (\mathbf{x}_i^\top, \mathbf{y}_i), i = 1, 2, \cdots, n\}$,where $\mathbf{x}_i^\top \in \mathscr{X}, \mathbf{y}_i \in \mathscr{Y}$, the function of interest is denoted by f, V : the number of fold, n: sample size.

Output: Cross Validation score $CV(\hat{f})$

for *v=1 to V* **do**

Extract the validation set
$\mathscr{D}_v = \mathbf{z}_i \in \mathscr{D}_n : i \in [1 + (v-1) \times m, v \times m]$
Extract the training set $\mathscr{D}_v^c := \mathscr{D}_n / \mathscr{D}_v$
Build the estimator $\hat{f}^{(-\mathscr{D}_v)}(\cdot)$ using $\mathscr{D}_v^c$
Compute predictions $\hat{f}^{(-\mathscr{D}_v)}(\mathbf{x}_i)$ for $\mathbf{z} \in \mathscr{D}_v$
Compute the validation error for the vth chunk

$$\xi_v = \frac{1}{|\mathscr{D}_v|}\sum_{i=1}^{n} 1(z_i \in \mathscr{D}_v) L(y_i, \hat{f}^{(-\mathscr{D}_v)}(x_i))$$

Compute the output CV score

$$CV(\hat{f}) = \frac{1}{V}\sum_{v=1}^{V}\hat{\xi_v}$$

Cross validation is one of the main techniques we used in this research to tune the parameters for the machines. For instance, in figure 5.1, it shows tuning the parameters for the support vector machine with Gaussian kernel, we could directly visualize the best parameter in the plot, which is the lowest point.

Grid search: Grid Search is a naive but straightforward approach to trying every possible configuration. Since Deep Neural Networks have many parameters (for example: learning rate, dropout rate, batch size, etc.), the challenge of this approach is "the curse of dimensionality." This means that the more dimensions we add, the more the search will explode in time complexity and computing power. For example, we want to create four hidden layers Neural Networks model, and for each layer, we add a dropout rate, also add a batch size to tune, and each parameter we choose from 2 potential good candidates, then we will need to run $2^9 = 512$ times. Because of this limitation the dimensions used in the experiments are less than or equal to 4, and the number of neurons in each layer goes from 16 to 256 in this research. The R package "tfruns," is a suite of tools for tracking, visualizing, and managing TensorFlow training runs. Figure 5.4a shows the tuning error; we want two losses as close as it could be. Figure 5.2b shows the saved hyperparameters. Using this table, we could find the best combination of the hyperparameters.

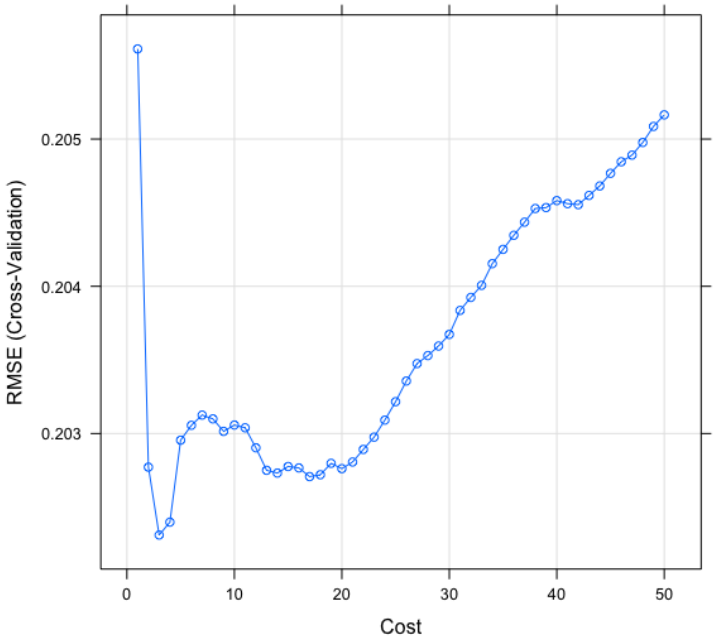
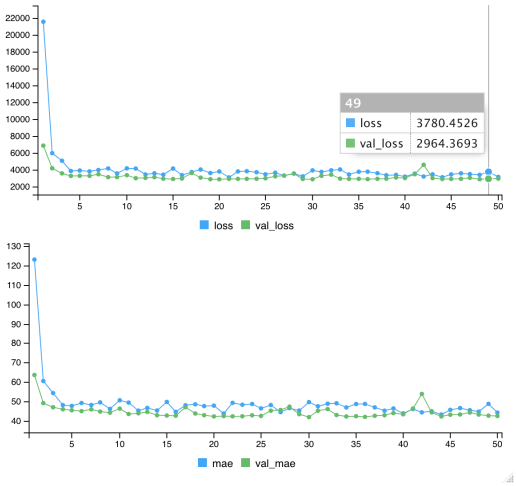


FIGURE 5.1: 5-fold cross validation to choose parameter for SVM with gaussian kernel



(A) DNN

	metric_accuracy	metric_val_accuracy	flag_dense_units1	flag_dense_units2	flag_dense_units3	flag_dense_units4	flag_dense_units5	flag_drop
1	0.6513	0.5146	128	64	64	16	8	
2	0.633	0.4951	256	64	64	16	8	
3	0.6343	0.5243	128	128	64	16	8	
4	0.6501	0.4563	256	128	64	16	8	
5	0.6403	0.4612	128	64	32	16	8	
6	0.6391	0.5146	256	64	32	16	8	
7	0.6476	0.4272	128	128	32	16	8	
8	0.6209	0.4854	256	128	32	16	8	
9	0.6561	0.5146	128	64	64	32	8	
10	0.6683	0.432	256	64	64	32	8	
11	0.6659	0.432	128	128	64	32	8	
12	0.6719	0.4709	256	128	64	32	8	
13	0.6306	0.466	128	64	32	32	8	
14	0.633	0.4806	256	64	32	32	8	
15	0.6428	0.4515	128	128	32	32	8	
16	0.6756	0.4854	256	128	32	32	8	
17	0.6403	0.4903	128	64	64	16	8	
18	0.644	0.4757	256	64	64	16	8	
19	0.6051	0.5388	128	128	64	16	8	
20	0.6245	0.5097	256	128	64	16	8	
21	0.6221	0.4806	128	64	32	16	8	
22	0.6075	0.5	256	64	32	16	8	
23	0.6282	0.4612	128	128	32	16	8	
24	0.6379	0.4854	256	128	32	16	8	
25	0.6464	0.3786	128	64	64	32	8	

(B) hyperparameters Table

FIGURE 5.2: Tuning hyperparameters for the Deep Neural Networks

5.2 Simulation Study

Simulation studies are the experiments that create data by pseudo-random sampling. A key advantage of simulation studies is that one could control the generated data. For example, one could change the parameters or control the sample size. By using the simulation data, we can have a better sense on how the machines are performing overall. In this section, we have regression and binary classification, two kinds of simulation studies.

5.2.1 Regression Example

In the regression simulation study, we generate 25 variables that are generated from a multi-normal distribution. The response variable is simply a linear combination of some of $\mathbf{X}$ and adds error ϵ which is generated from a normal distribution; the $\mathbf{Y}$ is given by

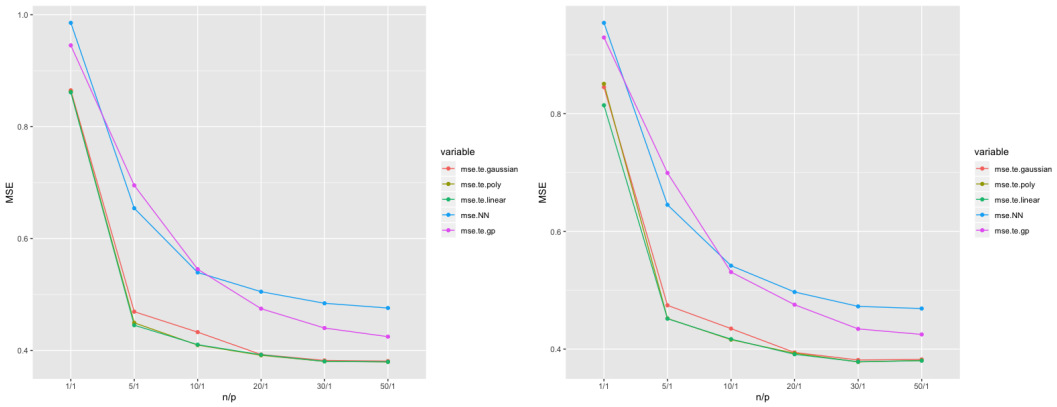
$$\mathbf{Y} = \mathbf{x}^\top \boldsymbol{\beta}^\star + \varepsilon \tag{5.2}$$

Let $\boldsymbol{\beta}^\star = (\beta_1, \beta_2, \cdots, \beta_p)^\top$. Let's consider an example where $\boldsymbol{\beta}^\star \in \mathbb{R}^p$, with $p = 25$, $\beta_j^\star = 0$ for $j \notin \{3, 5, 7, 11, 13, 17, 19\}$, except for $\beta_3^\star = 0.3$, $\beta_5^\star = 0.5$, $\beta_7^\star = -0.7$, $\beta_{11}^\star = 0.11$, $\beta_{13}^\star = 0.13$, $\beta_{17}^\star = 0.17$, $\beta_{19}^\star = 0.19$.

By changing the number of data points (table 5.1), we get six sets of data. We run five machines then compare the test error: (a) average test error; (b) median test error. Figure 5.3 shows the median and mean of 50 runs of the Mean Squared Error (MSE) on the test data; based on the plots, we could see all the methods improve tremendously when sample size n changes from 25 to 50, SVM with linear kernel, Gaussian kernel and polynomial kernel slightly improve after $n = 500$ compared to DNN and GP. All the SVM with kernel outperformed the DNN, GP is catching up with DNN at 250 data points, and starts to get better than DNN after that.

TABLE 5.1: Simulation data for regression Example

SN	partition	n	p	$\kappa = n/p$
1	regression-25	25	25	1/1
2	regression-125	125	25	5/1
3	regression-250	250	25	10/1
4	regression-500	500	25	20/1
5	regression-750	750	25	30/1
6	regression-1250	1250	25	50/1



(A) Mean MSE plot (B) Median MSE plot

FIGURE 5.3: These two plots are the simulation regression results for 5 machines. The left image shows the mean MSE in the 50 runs. The right image shows the median MSE in the 50 runs

5.2.2 Classification Example

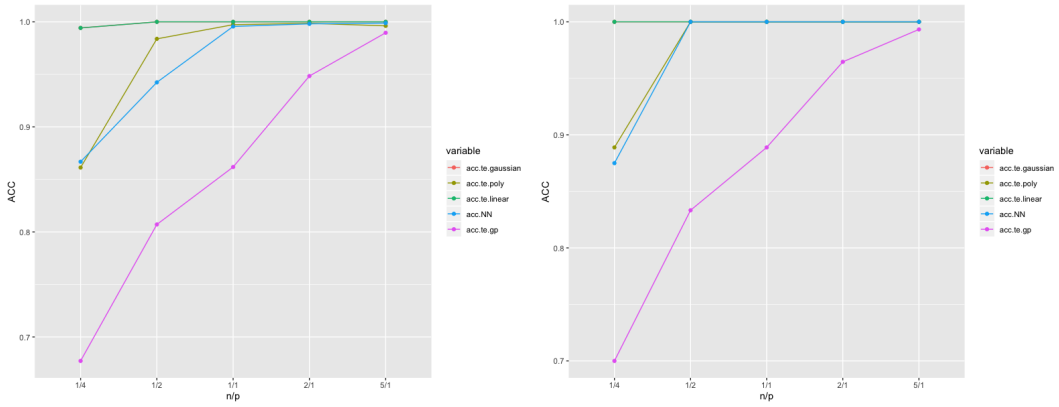
In the classification simulation study, we generate 100 variables. All variables are generated from a multivariate Gaussian distribution, and the response variable $\mathbf{Y}$ is generated from a Bernoulli distribution with the probability of success given by the function:

$$(\mathbf{Y}|\mathbf{x}) \sim \text{Bernoulli}(\pi(\mathbf{x})) \quad \text{where} \quad \pi(\mathbf{x}) = \Pr(\mathbf{Y} = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}^\top \beta}} \tag{5.3}$$

And changing the number of data points from 25 to 500, which will give us different values of $\kappa = \frac{1}{4}, \frac{1}{2}, \frac{1}{1}, \frac{2}{1}, \frac{5}{1}$. κ is an essential parameter in this thesis, and it shows the models' sensitiveness of the sample size. Table 5.2 shows the details of the data used in the simulation classification study. We compare the test error: Accuracy. We compare the median and mean of 50 runs of the accuracy on the test data. Figure 5.4 shows the results. Based on the plots, we could see most of the methods behaved much better when we increase the sample size from 25 to 50. SVM with linear kernel and Gaussian kernel could even reach 100 % accuracy nearly all the time. DNN is not a very competitive machine at first, but it starts to catch up when we increased the sample size to 100. SVM with linear kernel works very well on this data compared to the other machines. GP is not very competitive for this data set but eventually caught up when we increased the data to 500.

TABLE 5.2: Simulation data for classification example

SN	partition	n	p	$\kappa = n/p$
1	classification-25	25	100	1/4
2	classification-50	50	100	1/2
3	classification-100	100	100	1/1
4	classification-200	200	100	2/1
5	classification-500	500	100	5/1



(A) Mean ACC plot (B) Median ACC plot

FIGURE 5.4: These two plots are the simulation classification results for 5 machines.The left image shows the mean MSE in the 50 runs. The right image shows the median MSE in the 50 runs

In the simulation study, both regression results and classification results show that kernel machines have a better output than Deep Neural Networks most of the time. Compared to the kernel models, DNN's behavior depends more on the richness of the sample. In the next section, we present more results based on the real-world data.

5.3 Computational Explorations for real life data

For the real-world data, we used 10 regression datasets, and 16 classification data sets that included binary classification and multi-classification data. We thank Ogundepo and Fokoué, 2019 and MNIST (public data) for the contribution of the data sets. We are trying to cover as many areas as possible, for instance: medical, traffic, economic, etc. Table 5.3 and 5.4 briefly describe the data sets, the sample size, the number of response variables and $kappa = \frac{n}{p}$ which represents the informational richness. For the classification data there are three comparison groups: nine data sets for the binary classification, three data sets for the multi-classification, and four data sets for the parameter κ's influence.

TABLE 5.3: Regression Datasets table

SN	Dataset	n	p	$\kappa = n/p$
1	Airfoil	1503	6	250.5
2	Auto mpg	392	8	49
3	computer hardware	209	7	29.86
4	Concrete strength	1030	9	114.44
5	Diabetes	442	11	40.18
6	Ducan MBA	203	7	29
7	GPA	141	5	28.2
8	hprice	506	6	84.33
9	Istanbul stock	536	8	67
10	Mortality	992	4	233.75

5.4 Regression Results

For the regression study, there are ten data sets to be used to compare the five machines. The distribution of test errors of all the machines over 50 replications is shown in Figure 5.3. Among the ten boxplots, we could see the SVM with kernels are mostly better than DNN and have a smaller range compared to DNN. DNN only outperformed GP once (the 3rd boxplot). DNN did slightly better than SVM with the polynomial kernel. In the regression comparison, we would evaluate minimum, median, maximum and mean of the test error which is shown in Table 5.5, 5.6, 5.8 and 5.7. Among all the tables, SVM with Gaussian kernel wins seven times in the minimum table, six times in the median table, seven times in the mean table, and seven times in the maximum table, which makes SVM with Gaussian kernel the winner among the five learning machines. GP wins two times in the median table and maximum table, and one time in the mean table. DNN only wins two times in the minimum table, and there is a tie with the SVM with linear kernel.

TABLE 5.4: Classification Datasets table

SN	Dataset	n	p	Number of classes	$\kappa = n/p$
1	Asthmatic	405	11	2	36.82
2	Breast cancer	569	10	2	56.90
3	Congressional voting	435	17	2	25.59
4	Cryotherapy	90	7	2	12.86
5	Social network	400	4	2	100
6	Gender voice	3168	21	2	150.86
7	Diabetic	1151	20	2	57.55
8	Sonar	208	61	2	3.41
9	Indian Liver Patien	538	11	2	53
10	Balance scale	625	5	3	125
11	Cars	1728	7	4	246.86
12	Seeds	210	8	3	26.25
13	MNIST-112	112	784	10	1/7
14	MNIST-300	300	784	10	0.38(appro 3/7)
15	MNIST-784	784	784	10	1
16	MNIST-1586	1586	784	10	2

TABLE 5.5: Regression minimum MSE table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
Airfoil	0.335	0.378	0.370	**0.253**	0.346
Auto mpg	**2.33**	2.50	2.75	2.65	2.38
computer hardware	**28.41**	31.55	31.58	31.71	33.89
Concrete.	**5.65**	6.09	9.56	6.72	6.45
Diabetes	49.66	50.33	**49.58**	50.28	50.59
Ducan MBA	0.168	0.177	**0.166**	**0.166**	0.174
GPA	**0.265**	0.270	0.272	0.350	0.271
hprice	**2839.00**	3075.46	4311.29	3613.33	3709.67
Istanbul stock	**0.0124**	**0.0124**	**0.0124**	0.0137	0.0143
Mortality	**0.00008**	**0.00008**	0.00045	0.00225	0.00012

TABLE 5.6: Regression mediam MSE table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
Airfoil	0.49	0.53	0.53	0.54	**0.47**
Auto mpg	**2.84**	3.01	3.43	3.47	2.97
computer hardware	**55.37**	73.00	68.19	64.22	124.89
Concrete	**6.39**	7.89	10.82	8.87	7.23
Diabetes	**55.42**	56.48	56.32	58.87	56.30
Ducan MBA	**0.21**	**0.21**	**0.21**	0.36	0.22
GPA	0.36	**0.35**	**0.35**	0.49	0.37
hprice	5077.29	5058.32	5631.73	5291.71	**4966.83**
Istanbul stock	**0.01**	**0.01**	**0.01**	0.02	0.02
Mortality	10.12	9.70	**9.16**	12.18	9.66

TABLE 5.7: Regression mean MSE table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
Airfoil	0.49	0.54	0.55	0.53	**0.47**
Auto mpg	**2.89**	3.03	3.44	3.80	2.95
computer hardware	**58.83**	79.93	71.27	68.96	127.09
Concrete	**6.42**	7.61	10.81	9.25	7.25
Diabetes	**55.18**	55.92	55.98	59.19	56.32
Ducan MBA	**0.21**	0.22	**0.21**	0.37	0.22
GPA	0.36	**0.35**	**0.35**	0.53	0.36
hprice	**4928.90**	4893.10	5577.50	5155.59	4989.48
Istanbul stock	**0.01**	**0.01**	**0.01**	0.02	0.02
Mortality	9.50	8.76	**8.30**	12.03	8.89

TABLE 5.8: Regression maximum MSE table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
Airfoil	0.67	0.75	0.75	0.74	**0.66**
Auto mpg	**3.69**	3.88	4.22	6.50	4.06
computer hardware	**112.15**	156.68	130.46	119.84	222.77
Concrete	**7.72**	8.70	12.73	16.26	8.21
Diabetes	**61.15**	62.05	62.05	80.75	61.89
Ducan MBA	0.29	0.29	**0.25**	0.77	0.26
GPA	**0.44**	0.46	0.45	0.97	0.43
hprice	**5907.38**	5918.57	6703.37	6278.60	5982.95
Istanbul stock	**0.02**	**0.02**	**0.02**	0.03	**0.02**
Mortality	18.95	18.64	**12.86**	39.97	13.33

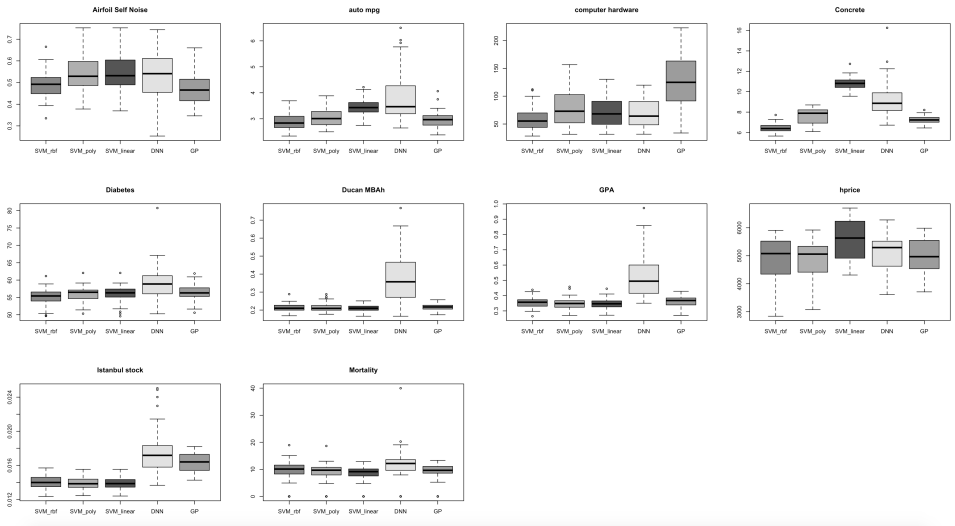


FIGURE 5.5: Comparison boxplots of MSE for 10 regression data sets

TABLE 5.9: Binary Classification maximum accuracy results table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
Asthmatic	**0.913**	0.898	**0.913**	0.891	0.896
breast cancer	**0.988**	0.981	0.981	0.987	0.982
congressional voting	**0.992**	**0.992**	**0.992**	0.986	0.976
cryotherapy	0.935	0.969	0.960	**1.000**	**1.000**
Social network	0.954	**0.969**	0.908	0.962	0.956
gender voice	0.984	**0.992**	0.981	0.981	0.984
Diabetic	0.785	0.787	**0.788**	0.771	0.764
Sonar	**0.948**	0.944	0.865	0.865	0.900
Indian Liver Patient	0.753	0.781	0.781	0.750	**0.803**

TABLE 5.10: Binary Classification median accuracy results table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
Asthmatic	0.837	0.838	**0.845**	0.823	0.830
breast cancer	0.961	**0.962**	0.958	0.957	0.958
congressional voting	**0.958**	0.955	0.957	0.953	0.947
cryotherapy	0.849	0.830	0.862	0.880	**0.895**
Social network	0.890	0.899	0.824	0.891	**0.902**
gender voice	0.976	**0.981**	0.975	0.970	0.972
Diabetic	**0.742**	0.734	0.741	0.707	0.692
Sonar	0.817	**0.857**	0.764	0.771	0.789
Indian Liver Patien	0.698	0.705	0.703	0.697	**0.708**

5.5 Classification Results

5.5.1 Binary Classification Results

In this section, we test five machines on nine binary classification datasets, we compare the minimum, median, maximum and mean of the test error which shows in table 5.12, 5.10, 5.11 and 5.9. We also generate distributions of the test error over 50 replications, which is shown in Figure 5.6. The boxplots show that DNN is comparable with other machines. In the binary classification test, DNN performs better than in the regression test and has a relatively smaller range of test error. In the maximum comparison table, SVM-RBF wins four times; SVM-poly and SVM-linear each win three times; GP wins two times, and DNN wins one time, but it ties with GP in this time. In the medium comparison table, SVM-poly and GP each win three times SVM-RBF wins two times, SVM-linear wins one time, DNN wins 0 times. SVM-poly wins four times in the mean table, DNN once but tie with SVM-poly and SVM-RBF in this one time, GP win three times in the mean table. For the minimum table, DNN only wins once, SVM-poly wins five times. So based on all the binary classification results, SVM-poly is the winner among the five machines; DNN does not outperform other machines. Even though the Kernel machines still seem to give a better predictive performance based on the tables and the boxplot, DNN is still comparable good with other machines in these cases.

TABLE 5.11: Binary Classification mean accuracy results table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
Asthmatic	0.843	0.840	**0.846**	0.822	0.830
breast cancer	**0.959**	**0.959**	0.958	0.957	0.956
congressional voting	**0.956**	**0.956**	0.955	**0.956**	0.945
cryotherapy	0.844	0.832	0.855	0.876	**0.893**
Social network	0.894	0.900	0.834	0.898	**0.901**
gender voice	0.976	**0.981**	0.974	0.971	0.972
Diabetic	**0.745**	0.734	0.744	0.708	0.695
Sonar	0.816	**0.854**	0.760	0.760	0.788
Indian Liver Patien	0.697	0.714	0.712	0.697	**0.710**

TABLE 5.12: Binary Classification minimum accuracy results table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
Asthmatic	0.765	0.755	**0.775**	0.716	0.745
breast cancer	0.912	**0.916**	0.906	0.912	0.894
congressional voting	0.917	0.915	**0.919**	0.907	0.911
cryotherapy	0.724	0.571	0.733	0.720	**0.800**
Social network	0.815	0.823	0.773	**0.846**	0.831
gender voice	0.962	**0.974**	0.962	0.959	0.964
Diabetic	0.682	**0.692**	0.675	0.618	0.637
Sonar	0.690	**0.761**	0.636	0.549	0.662
Indian Liver Patient	0.640	**0.658**	**0.658**	0.572	0.629

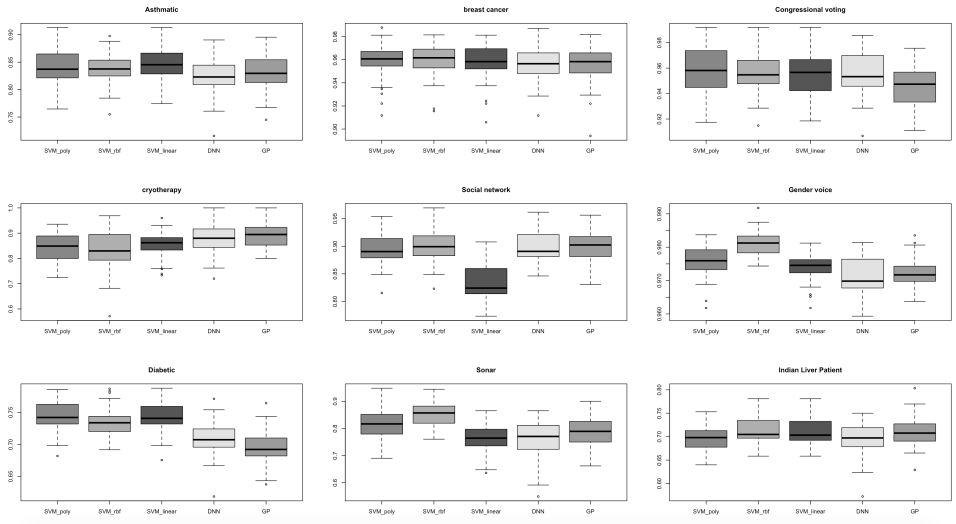


FIGURE 5.6: Comparison boxplot of accuracy for 9 binary classification data sets

TABLE 5.13: Multi Classification maximum accuracy results table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
balance scale	**1.00**	0.99	0.97	0.98	0.94
cars	**1.00**	0.99	0.88	0.98	0.91
seeds	0.99	0.99	**1.00**	0.97	0.99

5.5.2 Multi Classification Results

In the multi-class cases, we use seven datasets, divided into two sections. The first section contains three datasets, and the second section contains four datasets that explicitly test the κ effect of the prediction performance.

In the first section, we use three datasets to compare the minimum, median, maximum, and mean of the test error. Table 5.16 5.14 5.15 and 5.13 show the result. SVM-RBF wins two times out of three in the maximum and minimum comparison table and three times in the medium and mean comparison table, making it the winner in the multi-classification test. DNN and GP do not win even one time among the four comparison tables. However, based on the value of the 4 ACC tables, most machines have an above 95% ACC value, which is an excellent result for the multi-classification problem, and SVM-RBF has a couple 100% accuracy which makes this machine excellent for these data sets.

TABLE 5.14: Multi Classification median accuracy results table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
balance scale	**1.00**	0.96	0.92	0.96	0.91
cars	**0.98**	**0.98**	0.85	0.96	0.89
seeds	**0.95**	0.93	**0.95**	0.93	0.93

TABLE 5.15: Multi Classification mean accuracy results table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
balance scale	**0.99**	0.96	0.92	0.96	0.91
cars	**0.98**	**0.98**	0.85	0.96	0.89
seeds	**0.94**	0.92	**0.94**	0.92	0.92

TABLE 5.16: Multi Classification minimum accuracy results table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
balance scale	**0.95**	0.91	0.88	0.90	0.85
cars	**0.96**	**0.97**	0.82	0.94	0.85
seeds	0.80	**0.84**	0.82	0.83	0.82

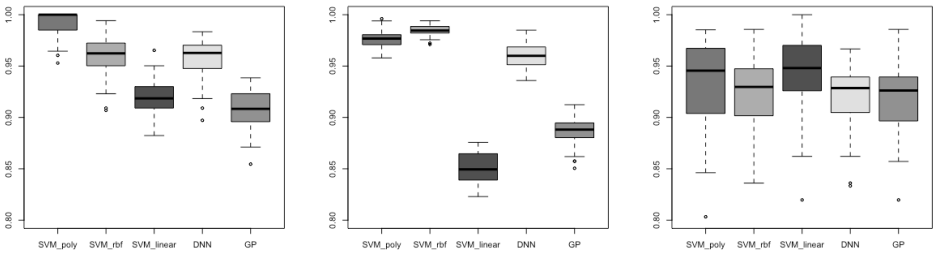


FIGURE 5.7: Multi Classification accuracy comparison boxplot

5.5.3 Predictive Performances on MNIST a function of κ

Although the current study suggests that the kernel machines and DNN models perform very similarly, the predictive performance of DNN seems more dependent on the richness of the data. In this section, we carry out an empirical study of the richness of the data ($\kappa = n/p$) affecting the performance.

The data we used in this section is MNIST data set. The MNIST databases mentioned earlier is a large database of handwritten digits from USPS. It has been commonly used for image recognition processing, and training and testing in the field of machine learning. Figure 5.8 shows 16 sample digits of MNIST data. We randomly take samples from the MNIST classification data set; we use 112, 300, 784, and 1568 samples as training data(table 5.17), and 100 samples as test data. We run each machine 50 times for each sample data and then compare the performance on test data.

TABLE 5.17: MNIST datasets

SN	partition	n	p	Number of classes	$\kappa = n/p$
1	MNIST-112	112	784	10	1/7
2	MNIST-300	300	784	10	3/7
3	MNIST-784	784	784	10	1
4	MNIST-1586	1586	784	10	2

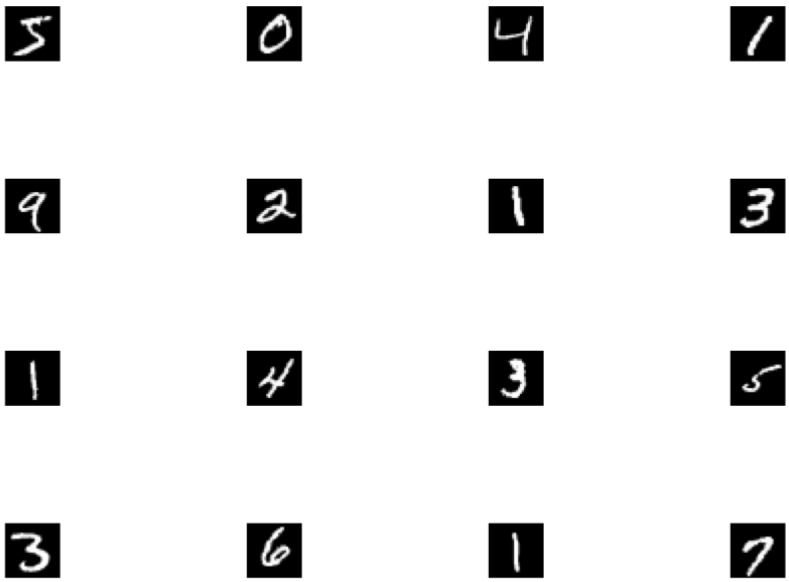
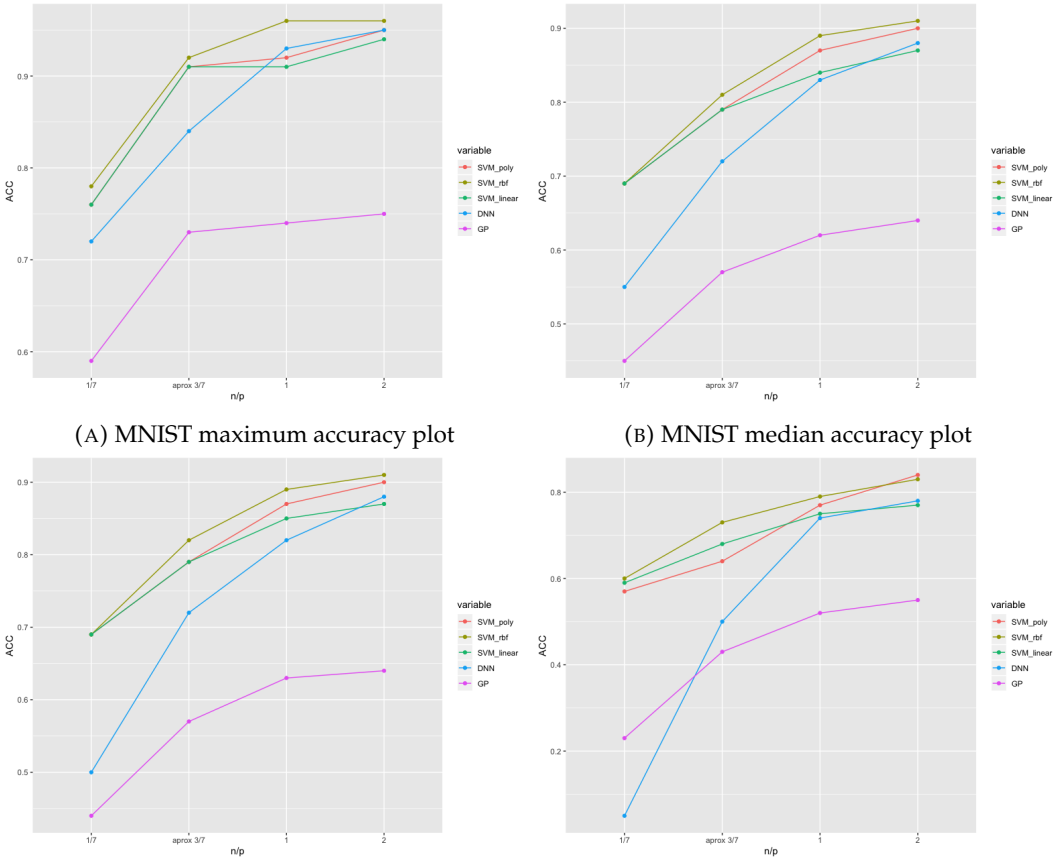


FIGURE 5.8: 16 sample digits from MNIST

We compare the minimum, mean, maximum, and medium of the accuracy results. In Figure 5.10, we could tell SVM-RBF, SVM-poly and SVM-linear are excellent machines for this task when using four different sizes of data. Next, let's look at the DNN's performance. From box plot one to box plot four we notice that as the sample sizes get larger, the range of DNN's accuracy gets smaller and catches up with SVM kernel machines. GP is falling behind all the times. In Figure 5.9, the performance of all the algorithms increases with more data. When using 112 data points which n/p is 1/7, the medium and maximum all start relatively low. When we increase n/p to approximately 3/7, all the machines have their accuracy increase tremendously, but the SVM kernel machines are still a lot better than DNN. From 3/7 to 1, the accuracy improves for all machines by a lot, but DNN model increases more than other models, and it becomes more compatible with the SVM kernel machines. When increasing n/p to 2, DNN's output is almost as good as SVM-RBF, but the tuning time of the DNN model is a lot longer than SVM-RBF machine. So based on these results, we could say that DNN is a greedy machine. It needs a rich data set to do the job.



(A) MNIST maximum accuracy plot

(B) MNIST median accuracy plot

(C) MNIST mean accuracy plot

(D) MNIST minimum accuracy plot

FIGURE 5.9: MNIST accuracy comparison plots

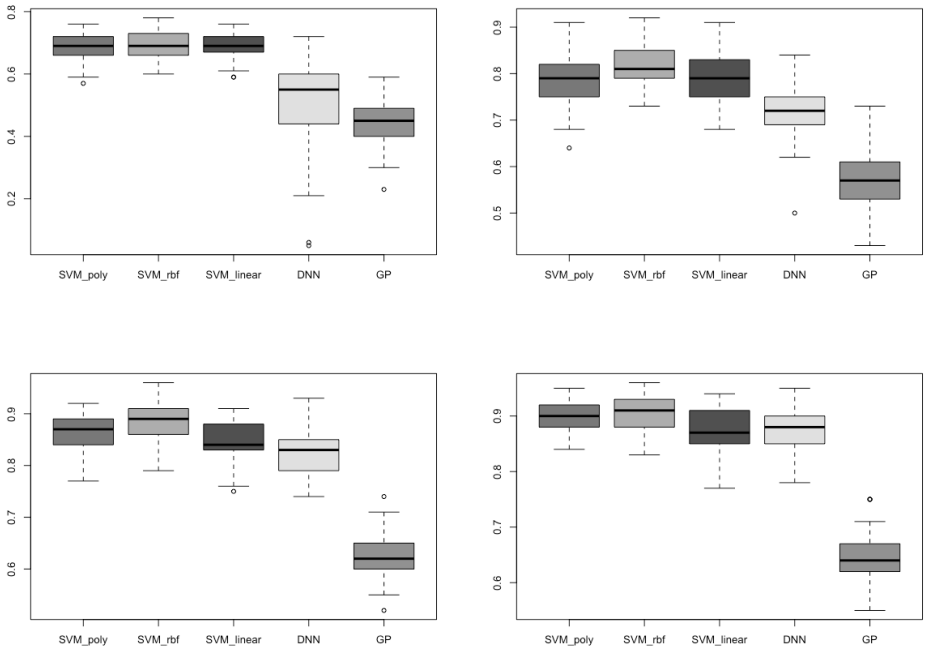


FIGURE 5.10: MNIST accuracy comparison boxplot

Chapter 6

Conclusion and Discussion

The whole purpose of this thesis was to explore the similarities and differences between kernel learning machines and Deep Neural Networks.

Chapter 4 provided a discussion of the similarities and differences between kernel learning machines and Neural Networks from a conceptual, methodological and theoretical perspectives. It was found that there are some similarities between the two learning paradigms, some of those similarities making the two paradigms almost identical at times. For instance, Gaussian processes turned out to be equivalent to single hidden layer neural networks with an infinite number of nodes in the hidden layer. Radial Basis Function Networks are formulated as Neural Networks with one single hidden layer and later shown to be kernel learning machines, with kernels of the specific type known as radial basis function kernels. In fact it turns out that for many versions of the so-called universal approximation theorem, single hidden layer neural networks are indistinguishable from kernel learning machines.

We later created objective experimental comparisons between Deep Neural Networks and Kernel machines in Chapter 5. We used many simulated and real-world datasets to compare kernel machines and Deep Neural Networks. Based on our experimental results, Kernel machines offer very competitive predictive performances and typically outperform Deep Neural Networks on almost all the data sets explored. Kernel machines also save resources compared to DNN according to the number of parameters and the tuning time. Deep Neural Networks turned out to be inordinately demanding in tuning time because of there complex internal structures, requiring vast amounts of time and computer resources.

$\kappa = n/p$ is one of the essential characteristic in this research, representing the richness of the data. Just as we anticipated, our results clearly show that the performance of DNN strongly depends on the richness of the data. In conclusion, DNN does not appear to be ideal for use if the dataset is small and limited. This is clearly more evidence for neural networks with more that one hidden layer. In the event of data poverty, i.e. when $\kappa < 1$, kernel methods appear to outperform DNN substantially. This alone shows that the claim that DNN should be a panacea used on every conceivable task, is at best an unsubstantiated claim.

Numerous articles based on the "No Free Lunch Theorem" have been published that compare the performance of various machine learning algorithms (Wolpert, Macready, et al., 1995;Wolpert and Macready, 1997; Ogundepo and Fokoué, 2019). The central point of the "No Free Lunch Theorem" is that the class of models for which a given learning technique is the adequate representation is limited. *In other words,no one technique will work well for all problems.* Even though DNN has been a

success in lots of very complex real life problems like self driving cars and complex image classification tasks, they also seem not to work well on very many tasks of importance to practitioners from all walks of life. Besides, our computational explorations did show that DNN do typically require relatively powerful computer systems, for some users, which may not be an ideal choice; due to the complex structure of DNN, which is also easier to lead to overfitting problems. DNN also required large amounts of data in order to yield adequate predictive performances comparable to those of kernel learning machines.

Our investigation found that kernel methods may be better suited for typical practical applications. A natural sequel to our investigation would be using more powerful computers to investigate how well DNN would perform with vast amounts of data.

Appendix A

Selected Code

A.1 Hyperparameters For Deep neural networks

```
  # Libraries
library(keras)
library(tensorflow)
library(tfruns)

# Data
data <-
str(data)
X_n<-
Y_n<-


# Matrix
data <- as.matrix(data)
dimnames(data) <- NULL

# Normalize
data[, 1:X_n] <- normalize(data[,1::X_n])
data[,Y_n] <- as.numeric(data[,Y_n]) -1

# Partition

ind <- sample(2, nrow(data), replace = T, prob = c(0.7, 0.3))
training <- data[ind == 1, 1:X_n]
test <- data[ind == 2, 1:X_n]
trainingtarget <- data[ind == 1, Y_n]
testtarget <- data[ind == 2,Y_n]

# One-hot encoding
trainLabels <- to_categorical(trainingtarget)
testLabels <- to_categorical(testtarget)

# Hyperparameter tuning
runs <- tuning_run("tuning.R",
                   flags = list(dense_units1 = c(32, 64),
                                dense_units2 = c(32, 64),
                                dropout1=c(0.1,0.2),
                                dropout2=c(0.1,0.2),
```

```
                                batch_size=c(32,64)))

# Best hyperparameter values
head(runs)


# Libraries
library(keras)
library(tensorflow)
library(tfruns)

# Data
data <-
str(data)
X_n<-
Y_n<-


# Matrix
data <- as.matrix(data)
dimnames(data) <- NULL

# Normalize
data[, 1:X_n] <- normalize(data[,1::X_n])
data[,Y_n] <- as.numeric(data[,Y_n]) -1

# Partition

ind <- sample(2, nrow(data), replace = T, prob = c(0.7, 0.3))
training <- data[ind == 1, 1:X_n]
test <- data[ind == 2, 1:X_n]
trainingtarget <- data[ind == 1, Y_n]
testtarget <- data[ind == 2,Y_n]

# One-hot encoding
trainLabels <- to_categorical(trainingtarget)
testLabels <- to_categorical(testtarget)

# Hyperparameter tuning
runs <- tuning_run("tuning.R",
                   flags = list(dense_units1 = c(32, 64),
                                dense_units2 = c(32, 64),
                                dropout1=c(0.1,0.2),
                                dropout2=c(0.1,0.2),
                                batch_size=c(32,64)))

# Best hyperparameter values
head(runs)
```

A.2 Tuning.R code

```
FLAGS <- flags(flag_integer('dense_units1', 32),
               flag_integer('dense_units2', 32),
               flag_numeric('dropout1', 0.1),
               flag_numeric('dropout2', 0.1),
               flag_integer('batcg_size',32))

# Model
model <- keras_model_sequential() \%>\%
  layer_dense(units = FLAGS$dense_units,
              activation = 'relu',
              input_shape = c(X-no)) \%>\%
  layer_dropout(rate = flags$dropout1) \%>\%
  layer_dense(units = FLAGS$dense_units,
              activation = 'relu') \%>\%
  layer_dropout(rate = flags$dropout2) \%>\%
  layer_dense(units = 3, activation = 'softmax')
# Compile
model \%>\%  compile(loss = 'categorical_crossentropy',
                  optimizer = 'adam',
                  metrics = 'accuracy')
# Fit
history <- model \%>\%
  fit(training,
      trainLabels,
      epochs = 50,
      batch_size = FLAGS\$batcg_size,
      validation_split = 0.2)
```

Bibliography

Alfaro-Almagro, Fidel et al. (2018). "Image processing and Quality Control for the first 10,000 brain imaging datasets from UK Biobank". In: *Neuroimage* 166, pp. 400–424.

Amarbayasgalan, Tsatsral, Bilguun Jargalsaikhan, and Keun Ho Ryu (2018). "Unsupervised Novelty Detection Using Deep Autoencoders with Density Based Clustering". In: *Applied Sciences* 8.9. ISSN: 2076-3417. DOI: 10.3390/app8091468. URL: https://www.mdpi.com/2076-3417/8/9/1468.

Barron, A. (1993). "Universal approximation bounds for superpositions of a sigmoidal function". In: *IEEE Trans. Inf. Theory* 39, pp. 930–945.

Barron, Andrew R. et al. (2008). "Approximation and learning by greedy algorithms". In: *The Annals of Statistics* 36.1, 64–94. ISSN: 0090-5364. DOI: 10.1214/009053607000000631. URL: http://dx.doi.org/10.1214/009053607000000631.

Baudat, G. and F. Anouar (2001). "Kernel-based methods and function approximation". In: *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*. Vol. 2, 1244–1249 vol.2. DOI: 10.1109/IJCNN.2001.939539.

Belkin, Mikhail, Siyuan Ma, and Soumik Mandal (2018). *To understand deep learning we need to understand kernel learning*. arXiv: 1802.01396 [stat.ML].

Cho, Youngmin and Lawrence Saul (2009). "Kernel Methods for Deep Learning". In: *Advances in Neural Information Processing Systems*. Ed. by Y. Bengio et al. Vol. 22. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2009/file/5751ec3e9a4feab575962e78e006250d-Paper.pdf.

Clarke, Bertrand, Ernest Fokoue, and Hao Helen Zhang (2009). *Principles and Theory for Data Mining and Machine Learning*. New York, NY : Springer-Verlag New York.

Cortes, Corinna and Vladimir Vapnik (1995). "Support-vector networks". In: *Machine Learning volume* 20, 273–297. DOI: https://doi.org/10.1007/BF00994018.

Craven, P. and G. Wahba (1979). "Smoothing noisy data with spline functions: Estimating the correct degree of smoothing by the method of GCV." In: *Numer. Math* 31, pp. 377–403.

Cybenko, G. (1988). *Continuous Valued Neural Networks with Two Hidden Layers are Sufficient*. Center for Supercomputing Research and Development. University of Illinois at Urbana-Champaign. URL: https://books.google.com/books?id=EppaHwAACAAJ.

— (Dec. 1989a). "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems (MCSS)* 2.4, pp. 303–314. ISSN: 0932-4194. DOI: 10.1007/BF02551274. URL: http://dx.doi.org/10.1007/BF02551274.

Cybenko, George (1989b). "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems (MCSS)* 2.4, pp. 303–314. ISSN: 0932-4194. DOI: 10.1007/BF02551274. URL: http://dx.doi.org/10.1007/BF02551274.

Dechter, Rina (1986). "Learning While Searching in Constraint-Satisfaction-Problems". In: *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*. AAAI'86. Philadelphia, Pennsylvania: AAAI Press, 178–183.

Drucker, Harris et al. (1997). "Support Vector Regression Machines". In: ed. by M. C. Mozer, M. I. Jordan, and T. Petsche, pp. 155–161. URL: http://papers.nips.cc/paper/1238-support-vector-regression-machines.pdf.

Fokoué, Ernest (2020). "Model Selection for Optimal Prediction in Statistical Machine Learning". In: *Notices of the American Mathematical Society* 67.2, pp. 155–168. DOI: DOI:https://doi.org/10.1090/noti2014.

Fokoué, Ernest and Boris Brimkov (2018). "The multifaceted impact of statistical methodology and theory in data science". In: *Mathematics for Applications* 7.1, pp. 1–2.

Fukushima, K. (2004). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36, pp. 193–202.

G., Wahba. (1998). *Support Vector Machines, Reproducing Kernel Hilbert Spaces, and the Randomized GACV*. Tech. Rep. 984. Dept. of Statistics, Univ. Wisconsin, Madison. URL: http://www.stat.wisc.edu/wahba.

G. Matthews, Alexander G. de et al. (2018). *Gaussian Process Behaviour in Wide Deep Neural Networks*. arXiv: 1804.11271 [stat.ML].

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. Adaptive computation and machine learning. MIT Press. ISBN: 9780262035613. URL: https://books.google.co.in/books?id=Np9SDQAAQBAJ.

Guimaraes Olinto, Gabriela and Ernest Fokoué (2018). "Kernelized Cost-Sensitive Listwise Ranking". In: *Mathematics for Applications* 7.1, pp. 31–40.

Hassoun, Mohamad H (1995). *Computational Capability of Artificial Neural Networks*. MIT Press, pp. 48–55. ISBN: 9780262082396.

He, Tong et al. (2020). "Deep neural networks and kernel regression achieve comparable accuracies for functional connectivity prediction of behavior and demographics". In: *NeuroImage* 206, p. 116276. ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2019.116276. URL: http://www.sciencedirect.com/science/article/pii/S1053811919308675.

Hornik, Kurt (1991). "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2, pp. 251 –257. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(91)90009-T. URL: http://www.sciencedirect.com/science/article/pii/089360809190009T.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5, pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208.

Jacot, Arthur, Franck Gabriel, and Clement Hongler (2018). "Neural Tangent Kernel: Convergence and Generalization in Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2018/file/5a4be1fa34e62bb8a6ec6b91d2462f5a-Paper.pdf.

Jacot, Arthur, Franck Gabriel, and Clément Hongler (2020). *Neural Tangent Kernel: Convergence and Generalization in Neural Networks*. arXiv: 1806.07572 [cs.LG].

Karatzoglou, Alexandros et al. (2004). "kernlab – An S4 Package for Kernel Methods in R". In: *Journal of Statistical Software* 11.9, pp. 1–20. URL: http://www.jstatsoft.org/v11/i09/.

Kawahara, Jeremy et al. (2017). "BrainNetCNN: Convolutional neural networks for brain networks; towards predicting neurodevelopment". In: *NeuroImage* 146, pp. 1038 –1049. ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.

2016.09.046. URL: http://www.sciencedirect.com/science/article/pii/S1053811916305237.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Kůrková, Věra (2002). "Universality and Complexity of Approximation of Multivariable Functions by Feedforward Networks". In: *Soft Computing and Industry: Recent Applications*. Ed. by Rajkumar Roy et al. London: Springer London, pp. 13–24. ISBN: 978-1-4471-0123-9. DOI: 10.1007/978-1-4471-0123-9_2. URL: https://doi.org/10.1007/978-1-4471-0123-9_2.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep Learning". In: *Nature* 521.7553, pp. 436–444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

Lee, Jaehoon et al. (2018). *Deep Neural Networks as Gaussian Processes*. arXiv: 1711.00165 [stat.ML].

Lee, Jaehoon et al. (2019). *Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent*. arXiv: 1902.06720 [stat.ML].

Liu, Weibo et al. (2017). "A survey of deep neural network architectures and their applications". In: *Neurocomputing* 234, pp. 11 –26. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2016.12.038. URL: http://www.sciencedirect.com/science/article/pii/S0925231216315533.

Murray, R.W. and E. Fokoué (2021). "Dropout Fails to Regularize Nonparametric Learners." In: *J Stat Theory Pract* 15.22. Featured article. DOI: DOI:https://doi.org/10.1007/s42519-020-00158-9. URL: https://doi.org/10.1007/s42519-020-00158-9.

Nair, Vinod and Geoffrey E Hinton (2010). "Rectified linear units improve restricted boltzmann machines". In: *ICML*.

Neal, Radford M. (1996). "Priors for Infinite Networks". In: *Bayesian Learning for Neural Networks*. New York, NY: Springer New York, pp. 29–53. ISBN: 978-1-4612-0745-0. DOI: 10.1007/978-1-4612-0745-0_2. URL: https://doi.org/10.1007/978-1-4612-0745-0_2.

Ogundepo, Ezekiel Adebayo and Ernest Fokoué (2019). "An Empirical Demonstration of the No Free Lunch Theorem". In: *Math. Appl.* 8, pp. 173–188.

Park, Jooyoung and Irwin W. Sandberg (Mar. 1993). "Approximation and Radial-Basis-Function Networks". In: *Neural Comput.* 5.2, 305–316. ISSN: 0899-7667. DOI: 10.1162/neco.1993.5.2.305. URL: https://doi.org/10.1162/neco.1993.5.2.305.

Rasmussen, Carl Edward and Christopher K. I. Williams (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press. ISBN: 026218253X.

Schmidhuber, Jürgen (2015). "Deep learning in neural networks: An overview". In: *Neural Networks* 61, 85–117. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2014.09.003. URL: http://dx.doi.org/10.1016/j.neunet.2014.09.003.

Scholkopf, Bernhard and Alexander J Smola (2018). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Adaptive Computation and Machine Learning series.

Snyder, David et al. (2017). "Deep Neural Network Embeddings for Text-Independent Speaker Verification." In: *Interspeech*, pp. 999–1003.

Tian, Yuchi et al. (2018). "Deeptest: Automated testing of deep-neural-network-driven autonomous cars". In: *Proceedings of the 40th international conference on software engineering*, pp. 303–314.

Wahba, G. (1990). *Spline Models for Observational Data*. Vol. 59. Philadelphia: SIAM CBMS-NSF Regional Conference Series.

— (2000). *An Introduction to Model Building with Reproducing Kernel Hilbert Spaces*. Tech. Rep. 1020. Dept. of Statistics, Univ. Wisconsin, Madison. URL: http://www.stat.wisc.edu/wahba.

Wolpert, D. H. and W. G. Macready (Apr. 1997). "No Free Lunch Theorems for Optimization". In: *Trans. Evol. Comp* 1.1, 67–82. ISSN: 1089-778X. DOI: 10.1109/4235.585893. URL: https://doi.org/10.1109/4235.585893.

Wolpert, David H, William G Macready, et al. (1995). *No free lunch theorems for search*. Tech. rep. Technical Report SFI-TR-95-02-010, Santa Fe Institute.

Wu, Qiuyi, Ernest Fokoué, and Dhireesha Kudithipudi (2018). "An Ensemble Learning Approach to the Predictive Stability of Echo State Networks". In: *Journal of Informatics and Mathematical Sciences* 10, pp. 1–15. ISSN: 0975-5748. URL: https://www.rgnpublications.com/journals/index.php/jims/article/view/827.

Zhang, Chiyuan et al. (2021). "Understanding Deep Learning (Still) Requires Rethinking Generalization". In: *Commun. ACM* 64.3, 107–115. ISSN: 0001-0782. DOI: 10.1145/3446776. URL: https://doi.org/10.1145/3446776.